Capa de comunicación JNI entre el sistema de seguimiento OpenCV y sistemas de razonamiento con ontologías.

Autor: Miguel Ángel Serrano Mateos
Director: Miguel Ángel Patricio
Codirector: Juan Gómez-Romero
Titulacion academica: Ingeniería en Informática

Madrid, Septiembre de 2009

AGRADECIMIENTOS

A mi familia que nunca dejó de ayudarme en mis estudios y en mi vida.

A los amigos que conocí en la universidad, en especial a Jesús, Tomás y Jacinto con los que pase mis mejores momentos.

A mi gran amigo Aldo que me enseña todos los días cosas nuevas.

A mis tutores por su paciencia y comprensión.

A mi ínclita y nunca bien ponderada novia, Yurika, sin la cual no habría podido hacer los encabezados de sección.

A todos aquellos que no he mencionado y cuyo apoyo ha sido fundamental.

INDICE

ACRÓNIMOS Y TÉRMINOS

.NET Framework: Es el modelo de programación de código administrado de Microsoft para la creación de aplicaciones en clientes de Windows, servidores y dispositivos móviles o incrustados.

API: Acrónimo inglés de Application Programming Interface. Consiste en un conjunto de funciones o llamadas al sistema, o bien, un conjunto de clases cuyo propósito es ofrecer un acceso a los servicios del sistema.

Blob: Es un conjunto de pixeles detectados que conforman una región concreta.

Boosting: Es una potente herramienta de aprendizaje, que proporciona una solución para las tareas de clasificación de aprendizaje supervisado. Los árboles de decisión son los clasificadores más populares en los esquemas de Boosting.

Códecs: Abreviatura de Codificador-Decodificador. Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de codificar un flujo de datos o señal y descifrarlo del mismo modo para tratarlo en un formato más apropiado para ciertas operaciones.

Combobox: Es una lista desplegable con entrada. Es un elemento de la interfaz gráfica de usuario que permite seleccionar una opción de una lista extensible de opciones.

DLL: Acrónimo inglés de Dynamic Linking Library. Son los archivos con código ejecutable, que se cargan bajo demanda de un programa por parte del sistema operativo Windows.

DOM: Acrónimo inglés de Document Object Model. Es una interfaz de programación de aplicaciones para los documentos XML.

EM: Acrónimo inglés de Expectation-Maximization. Se refiere a un algoritmo que estima los parámetros de una función de densidad de varias variables en una distribución Gaussiana.

Float: Es un término relativo a la programación y significa, tipo de datos en coma flotante. El coma flotante es un método de representación de números reales.

Frame: En castellano fotograma. Es una imagen particular dentro de una sucesión de imágenes que componen una animación.

Integer: Es un término relativo a la programación que significa tipo de datos enteros.

IPP: Acrónimo inglés de Integrated Performance Primitives. Rutinas optimizadas, desarrolladas por Intel, en las áreas de algoritmos de visión por computador. Se utilizan en los desarrollos de OpenCV.

Jace: Conjunto de librerías y programas que ayudan a escribir código en C++ que integra llamadas a métodos implementados en la máquina virtual de Java.

JDK: Acrónimo inglés de Java Development Kit. Consiste en un conjunto de herramientas, utilidades, documentación y ejemplos, para desarrollar aplicaciones Java.

JNI: Acrónimo inglés de Java Native Interface. Es una característica de la plataforma Java. Es una interfaz que permite a aplicaciones Java incorporar código de otro lenguaje y viceversa.

JVM: Acrónimo inglés de Java Virtual Machine. Es un programa nativo, diseñado para ejecutarse en una plataforma específica, capaz de interpretar y ejecutar instrucciones generadas por el compilador del lenguaje de programación Java.

Methontology: Es una herramienta que permite la construcción de ontologías a nivel de conocimiento.

MLL: Acrónimo inglés de Machine Learning Library. Es una sublibrería de OpenCV, se centra en patrones estadísticos de reconocimiento y agrupación y es muy útil para las tareas de visión.

nRQL: Acrónimo inglés de new Racer Query Languaje. Es un lenguaje de consultas utilizado por el razonador RACER que permite realizar consultas conjuntivas.

OpenCV: Abreviatura en inglés de Open Computer Vision. Es una biblioteca de funciones para la visión por computador en tiempo real. Puede ser usada libremente.

OWL: Acrónimo inglés de Ontology Web Language. Es un lenguaje de marcado, utilizado para compartir datos usando ontologías a través de Internet.

Path: Ruta de acceso a un recurso alojado en un computador.

PDF: Acrónimo inglés de Portable Document Format. Formato de documentos, ideado para su impresión, ya que específica toda la información necesaria para la presentación final.

Plug-in: En español su significado es complemento. Consiste en una aplicación que se relaciona con otra para aportarle nuevas y específicas funcionalidades. Esta aplicación interactúa por medio de la API y es ejecutada por la aplicación principal.

Random Trees: Es una técnica para la resolución de algoritmos de clasificación y regresión, que consiste en una colección de árboles de clasificación cuya salida es la clase más votada.

Scroll: Desplazamiento en 2 dimensiones. Incluye una barra de navegación para mover el contenido. Este movimiento, puede ser en sentido horizontal o vertical.

Tooltip: Es una herramienta de ayuda visual patentada por Microsoft. Funciona al situar o pulsar el ratón sobre un elemento, mostrando una ayuda acerca del elemento apuntado.

Track: También denominado pista. Es un bajo nivel de representación de una entidad en movimiento. Se percibe como una representación única o como un conjunto de ellas con propiedades de tamaño, color, velocidad… En este contexto, esta entidad es detectada por un tracker que contiene sus datos característicos.

Track Bar: Representa una barra de seguimiento estándar de Windows. Se desplaza en intervalos, con valores determinados.

Tracker: Programa software que ejecuta secuencialmente varios algoritmos de procesado de imágenes correspondientes a un vídeo, con la meta de seguir todos los objetivos desde el ámbito local.

Tracking: También denominado seguimiento. Consiste en establecer el número de objetos, con sus localizaciones, estados cinemáticos y demás características.

UC3M Tracker: Interfaz gráfica de usuario que inicia un sistema de seguimiento que incluye un enfoque cognitivo basado en ontologías.

URI: Acrónimo inglés de Uniform Resorce Identifier. Es una cadena de caracteres compacta para identificar un recurso físico o abstracto.

UTF-8: Acrónimo inglés de 8-bit Unicode Transformation Format. Es un formato de codificación de caracteres Unicode e ISO 10646 utilizando símbolos de longitud variable.

Windows Forms: Es el nombre dado a la interfaz de programación de aplicaciones gráficas, está incluido como parte de Microsoft .NET Framework.

World Wide Web Consortium: Abreviado W3C. Es un consorcio internacional que produce recomendaciones para la Wolrd Wide Web.

Wrapper: Es el programa que permite a ciertas clases trabajar juntas, cuando normalmente no podrían debido a incompatibilidad de interfaces. Traduce llamadas de una interfaz a llamadas de otra interfaz.

XML: Acrónimo inglés de Extensible Markup Language. Es un metalenguaje extensible de etiquetas. No es realmente un lenguaje en particular, sino un modo de definir lenguajes.

RESUMEN

Tradicionalmente las investigaciones en visión por computador han buscado enfoques basados en técnicas cuantitativas. Una nueva tecnología ha surgido basándose en el uso de la información de contexto con un potencial capaz de superar los métodos estadísticos en escenarios complejos.

Este trabajo integra tanto los enfoques cuantitativos como los contextuales a través de un sistema de comunicación, haciendo posible que la información en bruto obtenida a partir de las propuestas clásicas pueda modelarse y convertirse en información simbólica a partir de la cual razonar e inferir nueva información acerca de un entorno.

El objetivo final de la capa de comunicación es distribuir la información generada por cada una de las capas que representa los enfoques, complementando así la información necesaria en cada una de las capas y obteniendo una relación sinérgica entre ellas.

1 INTRODUCCIÓN

En esta primera sección, se va a realizar un resumen de los aspectos principales de este proyecto. En primer lugar, se indica el contexto y las necesidades que han llevado a la realización este trabajo; en segundo lugar, se explican cuáles son los objetivos que se han establecido para este caso; en tercer lugar se especifican los medios utilizados para su desarrollo y por último, se comentan las secciones en las que está dividida la memoria, junto a una somera descripción de su contenido.

1.1 AMBIENTACION Y NECESIDAD

Tradicionalmente, el análisis de vídeo por computación, se ha enfocado hacia técnicas cuantitativas, para calcular las propiedades y relaciones de todos los objetos aparecidos en la escena.

Uno de los problemas a los que se enfrenta la visión por computador, es la estimación del número de objetos en una escena y sus características físicas, como la posición o la velocidad de estos objetos, a este problema se le denomina seguimiento o tracking.

Esta cuestión, se ha tratado de abordar desde una perspectiva estrictamente numérica, estos intentos, han demostrado ser infructuosos cuando se trata con escenarios complejos en los que existe interacción entre diferentes objetos, por ejemplo una unión de elementos o una oclusión, modificaciones en los objetos y cambios en la escena. Es posible, en la mayoría de los casos, cometer un error en el etiquetado o en el tamaño del objeto para el que se hace el seguimiento. Una solución para superar estas dificultades es proporcionar al algoritmo de seguimiento información adicional acerca de las entidades observadas [1].

En la actualidad ha aumentado el interés por la representación simbólica, que permite realizar razonamientos con la información relativa al contexto. El contexto se define como cualquier conocimiento externo usado para completar los datos cuantitativos de una escena tratada con algoritmos de análisis de imagen.

La mayoría de las líneas de investigación en visión por computador, consideran el contexto desde un punto de vista local y cuantitativo, es decir el contexto de un objeto, es un cálculo sobre los píxeles de los objetos que lo rodea, según un criterio impuesto a priori, evidentemente esto es muy poco extensible, no se puede utilizar un mismo criterio para todas las escenas, sin embargo, los enfoques cognitivos construyen modelos simbólicos del mundo que observan y los expresan en un lenguaje formal, incluyendo el conocimiento preceptivo, es decir, los datos cuantitativos y el conocimiento textual.

Los lenguajes basados en lógica, llamados a liderar estos enfoques cognitivos, no han recibido demasiada atención a pesar de que son capaces de proporcionar una buena representación de los objetos de unas escena y poseen características de razonamiento. Esto se debe a que la utilización de la lógica de primer orden tiene ciertas desventajas, como las dificultades para hacerse extensiva y reutilizable o el hecho de que son semidecidibles.

Existe, sin embargo, algo capaz de superar estas dificultades, otorgándonos las propiedades deseadas, este avance, son las ontologías. Estas, son teorías axiomáticas de primer orden, que representan aspectos de una realidad, son creadas desde un punto de vista común y expresadas en un lenguaje formal.

1.2 OBJETIVOS

Este proyecto, trata de llevar a cabo un puente entre la percepción obtenida durante un seguimiento visión por computador y el conocimiento del contexto completando así un enfoque cognitivo.

El objetivo principal es integrar una herramienta que basándose en los resultados de un tracker clásico (previamente implementado), sea capaz de construir un modelo simbólico de la escena, a partir del cual operar, obtener información que, a priori, era desconocida y recomendar u ordenar modificaciones en el comportamiento del tracker.

Este modelo simbólico utilizado debe ser una representación ontológica de los objetos, relaciones y eventos ocurridos en la escena. La idea es que los objetos y relaciones, sean instanciados en cada frame del vídeo con los datos cuantitativos del algoritmo de tracking o seguimiento.

Las mejoras esperadas para esta herramienta al introducir los procesos de razonamiento son:

- Apoyo a la interpretación de los datos cuantitativos suministrados por el procedimiento de seguimiento, con el fin de construir un alto nivel de interpretación de la escena percibida [1].
- Detectar errores en el seguimiento, y proveer al tracker de esta información para modificar su comportamiento.

Adicionalmente, se ha pretendido dotar a esta herramienta de facilidades en su manejo para lo cual, se ha desarrollado una interfaz gráfica de usuario, que permite de forma independiente, ofrecer una mayor eficiencia y usabilidad al tracker resultante.

Algunas de las capacidades de esta interfaz deben ser:

- Independencia entre la interfaz, que actúa como traductor y el tracker que se ejecuta en un hilo distinto, posibilidad de realizar varias ejecuciones simultáneas, lo que permite realizar comparaciones entre los distintos parámetros ejecutados en tiempo real.
- Adaptabilidad, gracias a un sistema de lectura de su estructura, debe ser posible incluir nuevos controles que representen parámetros en la interfaz a partir de un fichero de configuración XML editable.
- Ahorro de tiempo, posibilidad de cargar y guardar archivos de datos, también estructurados en forma de XML, y cuyo contenido son los parámetros de entrada. Esto debe permitir utilizar ciertos archivos guardados con configuraciones en las cuales se ha invertido mucho tiempo, tratando de calibrar la aplicación para realizar el análisis de un de un vídeo determinado.
- Usabilidad, con esta herramienta, no tendría que ser necesario hacer una llamada con decenas de argumentos en una línea de comandos para iniciar el tracker, únicamente sería necesario cumplimentar los datos de la interfaz y ejecutar el análisis.

1.3 MEDIOS

Se exponen en este punto los desarrollos previos y los productos software usados para desarrollar y transmitir este trabajo.

1.3.1 LOS DESARROLLOS PREVIOS

Para actuar como enlace entre dos tecnologías como la visión por computador de corte clásico y los sistemas ontológicos, es necesario tener aplicaciones implementadas que representen ambos enfoques.

El producto utilizado para el caso de visión por computador, que actúa como base de todo el proyecto es un tracker de vídeo, también llamado, capa de tracking o capa de seguimiento, consiste en la una arquitectura de software con diferentes módulos de análisis de imágenes que son ejecutados de forma secuencial y que además, son capaces de cambiar el algoritmo utilizado sin afectar al resto de módulos. Estos módulos están dotados, por tanto, de tareas independientes, los hay que detectan objetos, otros estudian su trayectoria, etc.

Este tracker ha sido desarrollado por la Universidad Carlos III de Madrid y es uno de los proyectos llevados a cabo por el Grupo de Inteligencia Artificial Aplicada (GIAA).

La implementación del sistema ontológico, también denominado, capa de contexto, actúa como capa de alto nivel y utiliza procesos de razonamiento para interpretar los valores percibidos por el tracker. La estructura del modelo de representación basado en las ontologías, está organizado en niveles, de menos abstracto (información de cada pista o track), a más abstracto (descripción de la actividad desarrollada en el vídeo) [1], y cada uno de estos niveles se corresponde con una ontología.

Tanto la información observada por el tracker como la información de contexto está implementada en ontologías OWL a modo de modelo de conocimiento formal.

El estudio e implementación de los sistemas antológicos, es una nueva línea de investigación abierta en el Grupo de Inteligencia Artificial Aplicada (GIAA) de la Universidad Carlos III de Madrid.

1.3.2 LAS HERRAMIENTAS UTILIZADAS

Para la construcción de este proyecto, ha sido necesario hacer uso de varias herramientas, esto se debe a la diferencia de requisitos entre los múltiples elementos que constituyen la aplicación, a continuación, se explican el software utilizado y los productos resultantes asociados a estos.

- Microsoft Visual Studio, es un entorno de desarrollo integrado creado por Microsoft. Puede ser usado para crear aplicaciones de consola y aplicaciones de interfaz gráfica de usuario. Esta potente herramienta, es capaz de soportar cualquier lenguaje de programación, apoyándose en su editor y en su depurador.

  En esta ocasión, se ha utilizado tanto para la construcción de una interfaz gráfica de usuario, como para el desarrollo del tracker. Debido a que son productos independientes, estos han sido elaborados en proyectos distintos y con lenguajes diferentes, en el caso de la interfaz se ha utilizado C# para la implementación y XML para los archivos de configuración, para el tracker, sin embargo, se ha hecho uso exclusivo de C++.

  La versión del producto utilizada ha sido, Visual Studio 2005.

- En el marco de la comunicación entre la capa de seguimiento o tracker y la capa de contexto o sistema ontológico, se ha utilizado JNI (Java Native Interface). Esta, es una potente característica de la plataforma Java, que permite a los programadores, incorporen código nativo en lenguajes de programación como C, C++ o Java.

Para este proyecto, solucionar la comunicación entre las principales capas de la aplicación, consistía en levantar una máquina virtual de Java (JVM) durante la ejecución del código y tener la capacidad para realizar llamadas desde C++ (tracker) a las clases Java (ontologías).

- La visión por computador, utilizada en la capa de tracking o seguimiento, se ha basado en las librerías OpenCV (Open Computer Vision). Esta biblioteca libre, contiene funciones programadas, principalmente relacionadas con la visión por computador en tiempo real. Debido a que es multiplataforma puede ser usada en Mac OS X, Windows y Linux.

  El desarrollo de la capa de seguimiento, está íntimamente ligado a esta biblioteca y se ha usado fundamentalmente para el análisis de vídeo en tiempo real, lo que incluye las tareas relacionadas con el seguimiento.

  La versión utilizada en este caso ha sido OpenCV 1.0.

- En el plano del diseño de ontologías, se ha utilizado Protégé. Protégé es un editor de ontologías y una herramienta basada en el conocimiento. Es gratuita y de código abierto. La plataforma utiliza un editor OWL (Web Ontology Lenguaje), que es un lenguaje capaz de describir ontologías.

  Evidentemente, Protégé, se ha utilizado para definir las ontologías utilizadas en la capa de contexto.

  La versión usada en esta aplicación ha sido 3.4.1.

- Para la visualización de las ontologías, se ha utilizado el plugin de Protégé OWLViz. Este plugin permite la visualización y la navegación sobre las jerarquías de clases de las ontologías OWL, además es posible comparar la jerarquía de clases diseñada previamente, con la jerarquía de clases inferida durante el razonamiento.

  Esta aplicación añadida de Protégé se ha utilizado para facilitar la explicación sobre la experimentación del proyecto.

  La versión utilizada ha sido OWLViz 2.2.

- El plug-in OWLViz necesita de la aplicación GraphViz para poder funcionar. Este software es un visualizador de grafos de código abierto, capaz de realizar diagramas en varios formatos como imágenes, Postscript para su inclusión en PDF u otros documentos o visualizarlos en un navegador de grafos interactivo.

La versión utilizada en esta aplicación ha sido GraphViz 2.24.

- Para visualizar las ontologías se ha utilizado también el plugin de Protégé, denominado Jambalaya. Este ofrece vistas de los gráficos intercambiables, para una navegación interactiva de la ontología. Jambalaya es capaz de soportar una amplia gama de algoritmos de representación.

  Esta aplicación también se ha utilizado para facilitar la explicación sobre la experimentación del proyecto.

  La versión utilizada ha sido Jambalaya 2.7.0.

- La capa de contexto ha sido desarrollada íntegramente en Java. JDK (Java Development Kit) es un software que provee herramientas de desarrollo para la creación de programas en este lenguaje. JDK es de uso libre y gracias a su máquina virtual, es capaz de ejecutarse en cualquier entorno.

  El código relativo a la capa de contexto y las bibliotecas utilizadas, han sido compilados con esta herramienta. Para la ejecución de las clases resultantes, ha sido necesario levantar una máquina virtual procedente de este mismo producto.

  La versión utilizada en este caso ha sido JDK 1.5.0_19.

- En el ámbito de las pruebas se ha utilizado AVI Splitter, una utilidad para dividir y extraer cualquier fragmento de un vídeo AVI. El programa permite especificar el inicio y el fin del corte.

  Los videos utilizados en las pruebas han sido recortados con esta utilidad, de este modo se ha conseguido una longitud adecuada en tiempo.

  La versión utilizada ha sido AVI Splitter 2.11.

- El procesador de textos Microsoft Word ha sido utilizado para realizar la memoria de este proyecto de fin de carrera.

  En cuanto a la versión se ha utilizado tanto Microsoft Word 2003, como Microsoft Word 2007. (Debido a esto también se ha hecho uso del paquete de compatibilidad para formatos de archivo de Word, Excel y PowerPoint 2007 de Microsoft Office)

- El formato de documento portátil (PDF) ha sido utilizada para presentación del documento y la visualización de parte de la bibliografía.

  La versión utilizada ha sido Adobe Reader 9.1.0.

- PDFCreator es una aplicación que permite convertir documentos a formato PDF. El programa funciona bajo el sistema operativo Windows, aunque se distribuye bajo licencia de software libre.

  Los documentos diseñados en Microsoft Word han sido traducidos a PDF para su presentación definitiva, a través de esta herramienta.

  La versión de la aplicación utilizada ha sido PDF Creator 0.9.5.

- Adobe Photoshop, es una aplicación en forma de taller de pintura y está destinada al diseño gráfico a través del ordenador. Es capaz de soportar muchos tipos de archivos de imagen y puede utilizarse tanto en plataformas Mac OS como en Windows.

  Esta aplicación, ha sido utilizada para generar los diagramas y realizar los recortes necesarios de las imágenes expuestas.

  La versión utilizada ha sido Adobe Photoshop CS3.

1.4 ESTRUCTURA DE LA MEMORIA

Se describen en este apartado la estructura en capítulos de esta memoria y la variedad de sus contenidos.

Capítulo 1.0. Introducción

Es el punto actual, trata de explicar al lector los objetivos del proyecto, en el marco de las necesidades que lo han motivado.

Capítulo 2.0. Arquitectura general

Este capítulo presenta la estructura global y las características generales del producto en el que se enmarca el trabajo realizado para el proyecto.

Capítulo 3.0 Ontologías

Se realiza una explicación teórica acerca de esta tecnología y se profundiza en cómo y porque puede aplicarse a los entornos de análisis de vídeo para incrementar la calidad de estos productos.

Capítulo 4.0 Cadena de vídeo

Se explican en este punto la estructura, funcionamiento y características de tracker o cadena de vídeo. Es importante conocerlo en detalle, puesto que a partir de esta se han desarrollado el resto de productos por y para mejorar su rendimiento y funcionalidad.

Capítulo 5.0 Integración cadena de vídeo – ontologías

En este capítulo se realiza una explicación detallada del diseño e implementación de la comunicación entre el tracker y la cadena de vídeo, mediante la biblioteca JNI.

Capítulo 6.0 Experimentación

Se observan los resultados de la ejecución del sistema con un ejemplo del modelo simbólico generado y discuten y valoran los resultados obtenidos a través de las pruebas de tiempo realizadas.

Capítulo 7.0 Conclusiones y líneas futuras

Se describen los problemas durante la realización del proyecto y las conclusiones obtenidas a partir del trabajo realizado además de aportar nuevas ideas para futuros desarrollos.

Capítulo 8.0 Bibliografía

Se indica el manual bibliográfico utilizado necesitado para la realización del proyecto, lo que incluye manuales, publicaciones y recursos de Internet consultados.

Apéndice A

Este apartado trata sobre la interfaz gráfica desarrollada para la cadena de vídeo, incluye una explicación acerca de su instalación y desinstalación y un completo manual de usuario, en el que se describen su arquitectura, sus funcionalidades y un ejemplo de uso.

2 ARQUITECTURA GENERAL

Este capítulo trata de ofrecer una perspectiva general acerca de la estructura de esta nueva aplicación de tracking. Se explica en primer lugar, las partes de las que se compone la herramienta y su cometido en el entorno global, a continuación se explica su funcionamiento básico y para concluir se tratan las características principales de la aplicación.

2.1 ELEMENTOS DE LA ARQUITECTURA

La arquitectura general de esta herramienta de tracking de vídeo en tiempo real, está dividida en 2 capas separadas que se comunican y colaboran para mejorar el rendimiento de los análisis.

El primer elemento de la arquitectura, la capa general de tracking, ha sido durante los últimos años el núcleo de la visión por computador.

Este componente debe ser considerado en esta ocasión como un suministrador de información de bajo nivel. Esta información o percepciones se obtienen a partir de elementos que se introducen como entrada de esta capa:

- Imágenes procedentes de vídeo, las cuales sufren un análisis en varias fases.
- Tracks existentes de imágenes previas correspondientes al vídeo analizado.

Las percepciones obtenidas a partir de esta capa se proveen a capas superiores para un análisis más elaborado.

Las salida de esta capa, representa una de las salida de la aplicación y consiste en tracks o pistas que contienen diferente información acerca de lo objetos detectados en la escena. Estos tracks sufren cambios debidos a las recomendaciones realizadas por la capa superior de la arquitectura.

El otro componente de la arquitectura, es denominado capa de contexto. Esta capa ofrece un nuevo enfoque al campo del análisis de vídeo, otorgando capacidades de razonamiento lógico a través de conocimiento de alto nivel sobre la escena observada. Este modelo de procesado de contexto, se desarrolla como una capa superior del software de seguimiento o capa general de tracking.

La capa de contexto comienza su procesado cuando existen percepciones en la capa general de tracking que son suministradas, a través de la interfaz de comunicación que existe entre ambas.

Las entradas de este elemento son:

- El archivo de descripción del contexto, que contiene conocimiento adicional acerca de la escena y que está representado como descripciones de objetos estáticos, reglas de razonamiento, etc.
- Las percepciones enviadas en cada frame por la capa de bajo nivel, es decir, los valores calculados por el algoritmo de traking.

Una vez procesados los datos, la capa de contexto emite dos salidas, por un lado, envía recomendaciones sobre cómo tratar cada uno de los tracks hacia la capa de tracking, utilizando para ello la interfaz de comunicación y por otro lado emite un informe sobre la interpretación de la escena, por parte de la tecnología de alto nivel.

2.2 FUNCIONAMIENTO BÁSICO

Esta es una descripción resumida, del orden de funcionamiento general de esta aplicación.

1. La capa general de tracking, realiza una llamada diferente sobre los métodos de la interfaz de comunicación, dependiendo de si se crea, se actualiza o se eliminan los tracks.
2. La interfaz de comunicación, transforma la información cuantitativa a un modelo simbólico útil para la capa de contexto y actualiza la información visual elemental del modelo de instanciación actual de la capa de contexto.
3. Cuando una de las pistas es actualizada en la capa de contexto, se desencadenan varios procesos de razonamiento, que apoyados por el conocimiento de contexto, actualizan completamente la interpretación que existía previamente de la escena.
4. Las recomendaciones que serán enviadas a la capa de bajo nivel, estas son creadas a partir del razonamiento, el modelo de escena del momento y el conocimiento a priori, aportado por la descripción del contexto.
5. Para mantener la sincronía en la ejecución de las capas, es la capa general de tracking la que realiza la llamada a la interfaz de comunicación, para consultar las recomendaciones provenientes de la capa de contexto y convertirlas en acciones concretas sobre las pistas.

Este proceso de cinco pasos se repite en un bucle. Con cada frame del vídeo, la información saliente de la capa de tracking, modificada por las recomendaciones realizadas desde la capa de contexto, son realimentadas para realizar el análisis de la siguiente escena del vídeo.

El funcionamiento de ambas capas durante este proceso, es completamente independiente, si no se tiene en cuenta que el tracker activa la capa de contexto enviándole la nueva información correspondiente a cada nuevo frame y que ambas capas pueden quedar sincronizadas según el modo en el que se envíen las recomendaciones.

Este es un diagrama general del funcionamiento y la estructura de la aplicación de tracking de vídeo.

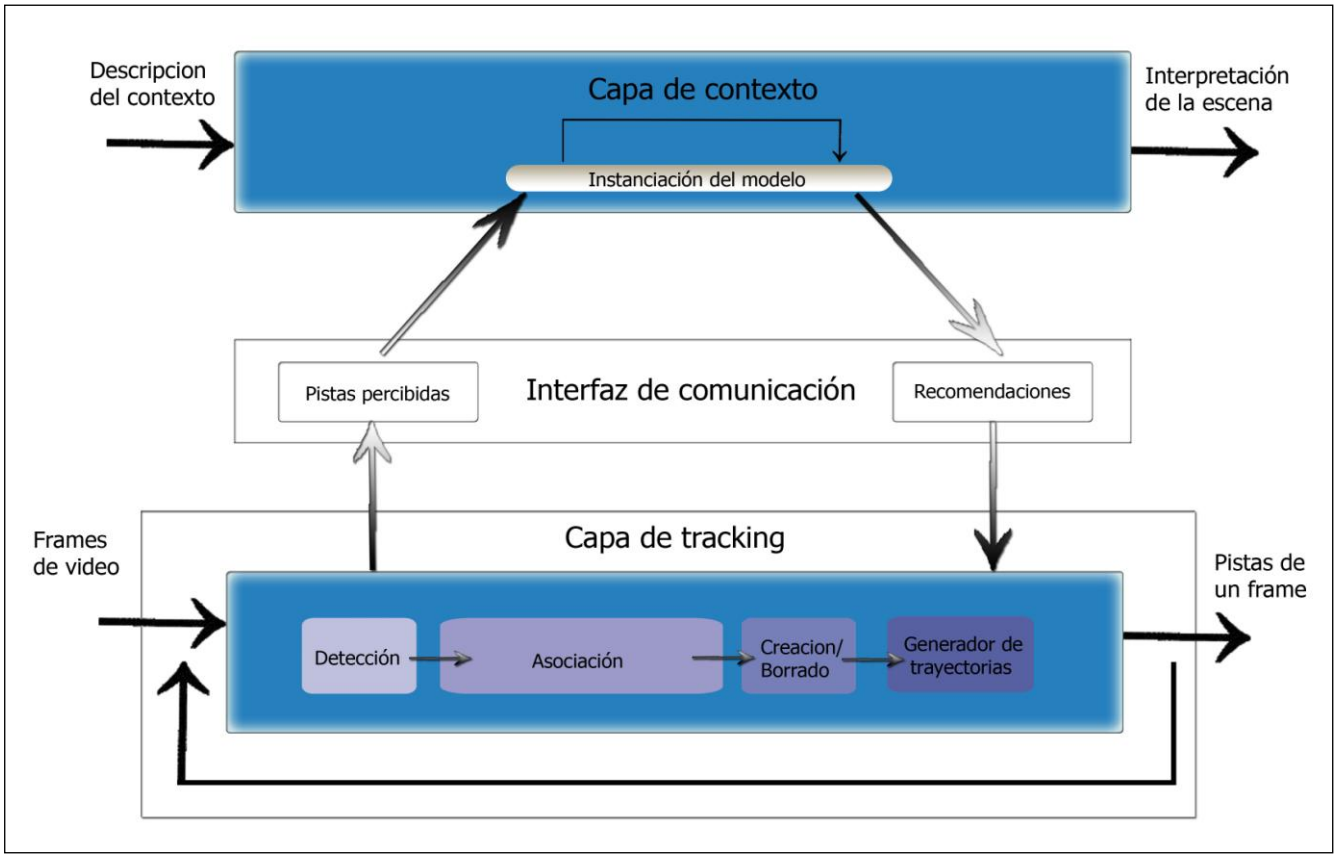

Ilustración 1

Quizás observando un ejemplo práctico sobre el procesado en la capa de contexto, puedan aclararse las dudas que no es capaz de despejar una explicación puramente teórica.

Supongamos que el tracker o capa de tracking, durante el análisis de una imagen de vídeo detecta una entidad que se mueve, es decir, una percepción, esta detección, forma parte del algoritmo de detección de tracks, que en ese momento se está ejecutando. La información del nuevo track detectado, debe ser enviada a la capa de contexto.

Si el objeto es nuevo, tiene un tamaño determinado y está cercano a un objeto conocido de la escena, la capa de contexto, debería deducir que el track detectado, una persona por ejemplo,

va a dejar de ser visualizada debido a ese objeto cercano, a esto se le denomina inferencia contextual y se calcula a través del conocimiento de la escena, ya sea implícito o explícito.

Finalmente tal y como se esperaba, unos frames más adelante en el vídeo, el track que representaba a este sujeto, ha desaparecido debido a que el objeto cercano de la escena lo ha tapado y por tanto debería ser eliminado del la capa de tracking.

La cuestión clave es la capacidad de la capa de contexto para recomendar al tracker, gracias a un razonamiento de alto nivel, que no elimine ese objeto, ya que realmente se mantiene en la escena aunque no esté visible, este track se mantendrá hasta que la persona deje de estar ocluida por el objeto y vuelva a visualizarse.

2.3 CARACTERÍSTICAS PRINCIPALES

Esta herramienta puede ser aplicada en diferentes aplicaciones de visión por computador, adaptando la arquitectura y extendiendo el modelo general con conocimiento específico del dominio.

Esta aplicación, resuelve varios retos que han sido destacados como cruciales en este campo [1]:

- Tener la capacidad de discernir, que conocimiento debe ser representado.
- Determinar cuáles son las representaciones formales de conocimiento adecuadas en cada caso.
- Aclarar como el conocimiento adquirido a través del análisis y el conocimiento contextual, se transforman de medidas numéricas a descripciones simbólicas.

Por todo ello este producto, se ha enfocado hacia los elementos que le otorgan estas características:

- La estructura y contenidos de del modelo de conocimiento.
- Las ontologías para codificar este conocimiento.
- Los procesos de inferencia para interpretar la escena en términos de descripciones simbólicas.
- Los procesos de inferencia para proporcionar información de retorno consistente en recomendaciones a los algoritmos de procesado de imágenes de bajo nivel.

3 ONTOLOGÍAS

Este capítulo trata de introducir al lector en el panorama de las ontologías con una breve introducción histórica; a continuación, se realiza una reflexión teórica acerca de esta tecnología. En tercer lugar se profundiza en la capa de contexto desde un punto de vista arquitectónico. Para concluir se explica el desarrollo de esta tecnología en el marco de la aplicación que se presenta.

3.1 INTRODUCCIÓN HISTÓRICA

Saber representar el conocimiento, es muy conveniente en el ámbito de la computación, con esta capacidad es posible realizar descripciones del mundo en un lenguaje formal.

El lenguaje tradicional para representar el conocimiento ha sido la lógica de predicados de primer orden. Los problemas de este uso tradicional se basan en el hecho de que es un proceso:

- Poco intuitivo.
- Tedioso.
- Semi-decidible, lo que significa, que no puede saberse si un algoritmo que determina si una expresión es cierta, terminará, para un caso en que esa expresión no pueda satisfacerse.

Con la intención de superar este problema de la semi-decibilidad, se suele trabajar con subconjuntos decidibles (completos) de la lógica de primer orden. La lógica descriptiva o lógica de descripciones, utiliza esta solución. Define varias lógicas con determinados niveles de expresividad que, dependiendo de los constructores primitivos que se empelen en cada representación de conocimiento, se garantizan o no una serie de propiedades computacionales [8].

Los constructores primitivos son utilizados para determinar qué propiedades se mantienen en el proceso de inferencia, delimitando los niveles de expresividad dentro de la lógica.

Por ejemplo, en el caso de la familia de lógicas attributive language, en su nivel básico se permiten conceptos atómicos, conceptos atómicos negados, intersección de conceptos, restricciones de valor, cuantificaciones existenciales limitadas y los conceptos de superior e ínfimo.

Quedarían en todo caso por resolver el hecho de que la construcción del conocimiento sea difícil de aprender y realmente poco atractiva de desarrollar. Las ontologías, son un

formalismo de representación del conocimiento capaz de abordar conjuntamente todos estos problemas. Tomando para ello la facilidad expresiva de los lenguajes basados en estructuras de red (redes semánticas y marcos) y la semántica formal basada en lógica de descripciones.

3.2 INTRODUCCIÓN TEÓRICA

La ontología es una ciencia que se preocupa de analizar las características de las cosas y las categorías a las que pertenecen. En el entorno de la ingeniería del conocimiento, esto significa que hace referencia al intento de formular un esquema conceptual en un dominio determinado. Se da por sentado, que este esquema tiene una comprensión compartida, es decir, ha sido consensuado proporciona un vocabulario común a los investigadores que necesitan compartir información acerca de un dominio y que el lenguaje utilizado puede procesarse mediante un computador.

Las ontologías son usadas habitualmente para capturar conocimiento sobre algún dominio, pero realmente existen muchas razones para crear una ontología:

- Compartir conocimiento común entre personas o agentes software.
- Reutilizar el conocimiento de dominio, lo que permite ahorrar esfuerzo en la adquisición y codificación de conocimiento.
- Obtener nuevos conocimientos a través del análisis.
- Separar el conocimiento del dominio del conocimiento operacional.
- Hacer explicitas las asunciones acerca de un dominio.

3.2.1 CONSTRUCCIÓN DE UNA ONTOLOGÍA

Una ontología describe conceptos básicos en un dominio y define las relaciones entre ellos. Los elementos básicos para su construcción son:

- Conceptos o clases: Son ideas básicas que determinan conjuntos de objetos del dominio. Un concepto de nivel superior generaliza a otro de nivel inferior. Esta idea es denominada TBox en lógica de descripciones.
- Individuos o instancias: Hechos concretos de un concepto.
- Relaciones o Roles o Slots: Son conexiones binarias entre individuos. Describe relaciones entre individuos o entre individuos y valores de un tipo de datos (entero, cadena de caracteres,...). Este mismo concepto es denominado Rbox en lógica de descripciones.

- Axiomas: Definen restricciones sobre los elementos que determinan las características y la forma en que pueden conectarse los elementos de una ontología. Este mismo concepto es denominado Abox en lógica de descripciones.

Una ontología junto con un conjunto de instancias individuales de clases constituye una base de conocimiento.

Crear una ontología es un proceso de refinamiento, que normalmente incluye

- Definir las clases.
- Organizar las clases en una jerarquía.
- Definir las relaciones y los valores permitidos.
- Rellenar los valores en las instancias.

Existen metodologías para realizar este proceso, se utilizan para crear ontologías a partir de cero, para la reutilización de ontologías tal y como están y para procesos de reingeniería de las mismas. Una de estas metodologías se llama Methontology, esta es una herramienta que permite la construcción de ontologías a nivel de conocimiento. La aplicación, consiste en desarrollar la ontología a partir de la identificación de las actividades principales, tales como, evaluación, configuración, conceptualización, integración de la implementación; un ciclo de vida basado en prototipos evolutivos y la especificación de los pasos para realizar las actividades, las técnicas, los resultados y su evaluación.

3.2.2 LÓGICA DE DESCRIPCIONES

Los lenguajes utilizados por las ontologías, son normalmente equivalentes a una lógica de descripciones decidible. La lógica de descripciones, es un formalismo para la representación del conocimiento estructurado que restringe la lógica de primer orden para asegurar una serie de propiedades computacionales [8].

Las primitivas de representación de la lógica de descripciones son las mismas que las de las ontologías:

- Los conceptos son equivalentes a TBox en lógica de descripciones.
- Las relaciones son equivalentes a RBox en lógica de descripciones.
- Los axiomas son equivalentes a Abox en lógica de descripciones.

La inclusión de constructores en una lógica, delimita la expresividad de la lógica.

Los constructores de una lógica, son sus características:

- Cuantificaciones existenciales limitadas.
- Conceptos atómicos negados.
- Conceptos de superior e ínfimo.
- Propiedades transitivas
- ...

La teoría de lógica de descripciones, determina que constructores pueden utilizarse si se desean ciertas propiedades, por ejemplo, decibilidad o eficiencia, del proceso de inferencia [8]. De este modo y según las propiedades que contengan aparecen las familias de lógicas.

3.2.3 LA REPRESENTACION DEL CONOCIMIENTO Y EL LENGUAJE OWL

El conocimiento, debe codificarse utilizando un lenguaje de representación, diferentes lenguajes ontológicos proporcionan diferentes facilidades, el más extendido es el Lenguaje de Ontologías Web (OWL, Ontology Web Language), surgido en el ámbito de la web semántica y recomendado como estándar por el World Wide Web Consortium (W3C).

Dependiendo de la expresividad de los lenguajes las representaciones ontológicas tienen diversas propiedades, por ello, las ontologías basadas en OWL pueden categorizarse en 3 subespecies:

- OWL Lite, es el lenguaje más simple en cuanto a sintaxis, es recomendable usarlo cuando se utilizan jerarquías de clasificación simple y solo son necesarias restricciones simples. La ventaja principal es que el razonamiento es más eficiente y resulta más sencillo trabajar con él.
- OWL DL, es mucho más expresivo que el anterior, está basado en la lógica de descripciones. Es un lenguaje completo, lo que significa que todas las conclusiones son computables. También es un lenguaje decidible por lo que todos los cálculos terminan en tiempo finito. Esta categoría, incluye todos los constructores pero solo pueden usarse bajo ciertas restricciones.

OWL-Full, es el sublenguaje OWL más expresivo a costa de perder la decidibilidad. Permite utilizar todos los constructores, en este caso, sin restricciones. Este subtipo de lenguaje, es interesante, cuando una alta expresividad, es más importante que garantizar si es decidible o completo.

3.2.4 RAZONAMIENTO SOBRE ONTOLOGÍAS

El razonamiento se define como la posibilidad de inferir conocimiento implícito a partir del conocimiento explícito. Sobre una base de conocimiento es el proceso que a partir de los axiomas del modelo (conocimiento explícito), se infieren nuevos axiomas (conocimiento implícito), mediante un procedimiento automático dictado por la lógica de descripciones.

La inferencia básica en los conceptos de una ontología, consiste en determinar si un concepto es satisfactorio, esto ocurre cuando no es contradictorio con el resto del conocimiento que abarca la ontología, otra tarea importante, es la inclusión de conceptos que consiste en determinar si un concepto es más general que otro. Cualquier razonamiento sobre conceptos (satisfacción, equivalencia o disyunción), según las propiedades de la lógica de descripciones puede reducirse a la inclusión de conceptos.

La inferencia elemental sobre individuos, consiste en determinar si existe, al menos, una interpretación conceptual (TBox) de una afirmación acerca de un individuo (ABox). Cualquier razonamiento sobre individuos puede reducirse a la comprobación de satisfacción o consistencia.

Una característica especial del razonamiento en lógica de descripciones es que suele mantenerse la hipótesis de mundo abierto. Esto significa que se supone que el conjunto de axiomas de una ABox no es completo. Por ejemplo, si en la Abox de una base de conocimiento se establece que Ángel y Juan son hijos de Pedro y no se dispone de más información al respecto en la Tbox, no puede inferirse que todos los hijos de Pedro son varones [8]. Por el contrario bajo la hipótesis de mundo cerrado esta inferencia podría efectuarse.

Las tareas de razonamiento sobre ontologías son llevadas a cabo por los motores de inferencia, estos son módulos software que implementan el algoritmo de pizarra, que se basan en la premisa de que si un concepto es más general que otro esto equivale a que es satisfactorio.

Existe una relación directa entre la expresividad de un lenguaje y la dificultad de razonar con él, de forma que cuanto más expresiva es una lógica, más complejo es el razonamiento.

En la actualidad hay varios motores de inferencia capaces de garantizar que el razonamiento terminara en tiempo exponencial. Destacan entre ellos:

- Racer: Acrónimo de Renamed Abox and Concept Expression Reasoner. Está basado en LISP. Proporciona facilidades para realizar tareas de razonamiento que incluyen procesamiento algebraico y cadenas de caracteres. La lógica que implementa es compatible con el lenguaje OWL DL, un nivel decidible del lenguaje OWL, además de

ofrecer un lenguaje para la consulta de instancias denominado nRQL. Actualmente, está mantenido en su versión comercial con el nombre RacerPro.

- FaCT: Acrónimo de Fast Classification of Terminologies. Está basado en C++. Implementa 2 razonadores asociados a lógicas distintas. Ofrece un lenguaje expresivo, al nivel de OWL DL. Soporta razonamiento con varias bases de conocimiento y utiliza los algoritmos de pizarra para desarrollar inferencia. Ofrece una versión cliente-servidor, basada en CORBA. La versión FaCT++ optimiza los algoritmos utilizados en FaCT.

- Pellet: Es un razonador OWL DL de código abierto, basado en Java. La API de Pellet proporciona funcionalidades para ver los tipos de validación y consistencia de las ontologías, la clasificación de las taxonomías, la verificación de las vinculaciones y responder a un subconjunto de las consultas de RDQL. Pellet está basado en los algoritmos de pizarra y es el primero que soporta la expresividad completa de OWL DL.

3.3 LA CAPA DE CONTEXTO

Como ya se ha visto en el punto 2 (Arquitectura general), la capa de contexto implementa procesos de razonamiento sobre los datos aportados por el análisis de la capa de seguimiento o tracking. El objetivo de esta capa es procesar la información de bajo nivel recibida y transformarla en recomendaciones entendibles por la capa de seguimiento. Para ello, es necesario que exista como entrada, un conocimiento de contexto, con el cual sea capaz de interpretar correctamente la escena.

Las dos entradas de la capa de contexto, tanto la percibida, capturada por el tracking como la contextual, están representadas con un modelo formal de conocimiento implementado como un conjunto de ontologías OWL.

3.3.1 LA ESTRUCTURA DE LA CAPA DE CONTEXTO

La estructura de este modelo de representación basado en ontologías, está organizado en varios niveles, del menos abstracto (información de un track), a más abstracto, (descripción de una actividad) [1]. Esta estructura ha sido diseñada de acuerdo al modelo JDL para la fusión de información y datos. JDL es un modelo de referencia que establece un vocabulario común para facilitar la comunicación y el entendimiento entre especialistas de fusión de información.

El modelo establece 4 niveles operacionales en la transformación de una entrada hasta que esté preparada para el nivel de conocimiento, estos niveles son: [1]

- Evaluación de la función de la señal, denotada como L0.

- Evaluación de la entidad, denotada como L1.
- Evaluación de la situación, denotada como L2.
- Evaluación del impacto, denotada como L3.
- Evaluación del proceso, denotada como L4.

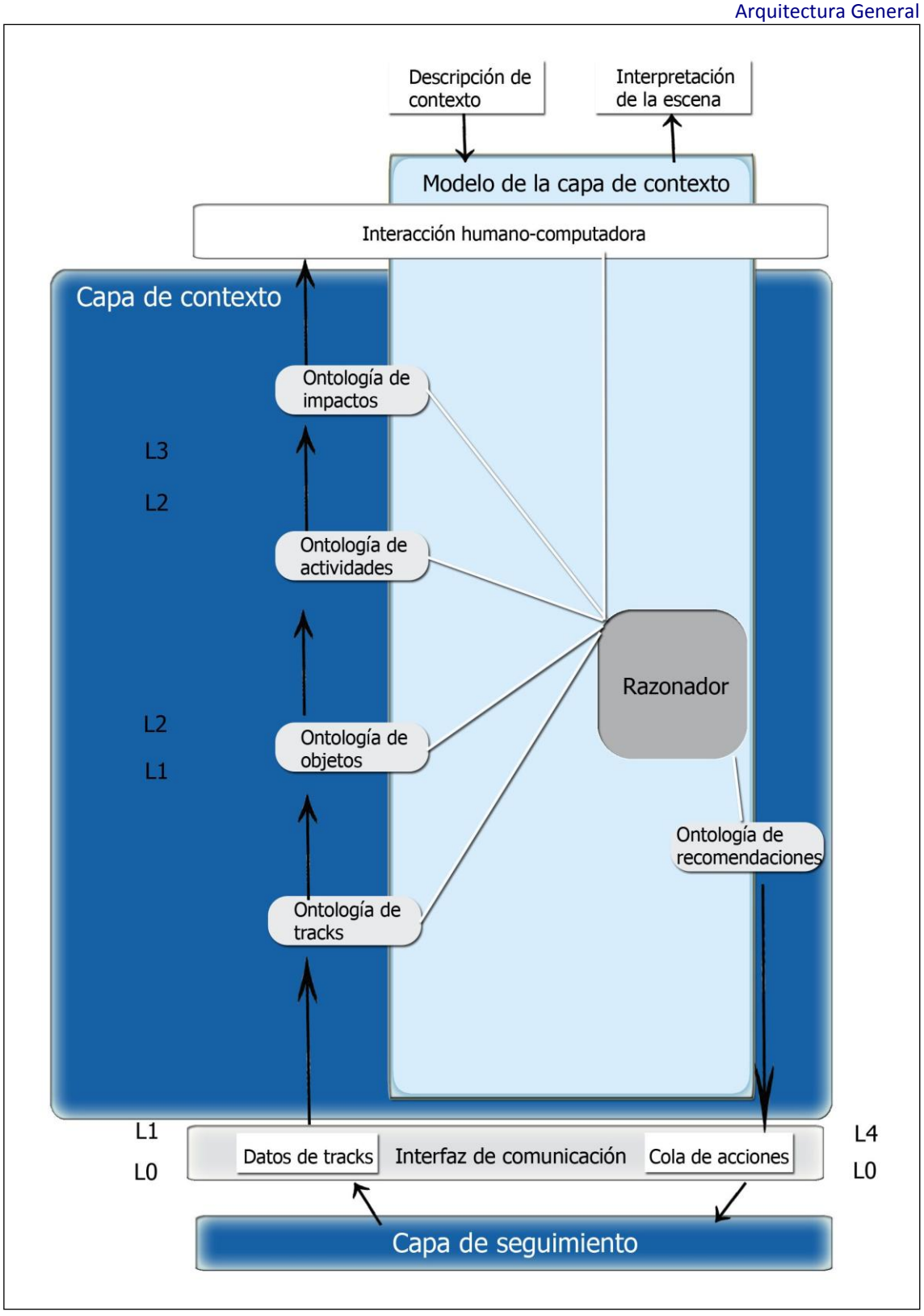

Ilustración 2

Dentro modelo de conocimiento de la capa de contexto, cada una de estas categorías se

correspondería los siguientes elementos del sistema de seguimiento:

- Información de las cámaras (L0): Esta información no es gestionada en la capa de contexto sino en la capa de seguimiento. La secuencia de vídeo grabada por las cámaras es la entrada que analiza la capa de seguimiento para obtener los tracks.
- Información de tracking (L1): Los datos de entrada de la capa de seguimiento, son considerados como los datos de salida de la herramienta de tracking Estos datos son la información de las entidades representados desde una perspectiva ontológica. Las instancias de los tracks y sus propiedades (posición, estado cinemático,...), frames,... son creados en esta representación inicial de los objetos móviles de la escena.
- Objetos de la escena (L1-L1/2): Los objetos de una escena son el resultado de la correspondencia entre los tracks y los objetos posibles. Estos objetos pueden contener elementos estáticos definidos a priori, puesto que ya han sido identificados como elementos con características propias. Sus propiedades y relaciones deben tenerse por tanto en cuenta, por ello existe este nivel intermedio entre L1 y L2.
- Actividades (L2): Las actividades son los comportamientos de los objetos de la escena. Las actividades describen relaciones entre objetos que duran en el tiempo como por ejemplo, la aproximación entre objetos o la agrupación de estos.
- Impactos y amenazas (L3): Las actividades pueden ser asociadas con un coste tras la evaluación. Esta capacidad es interesante para aplicaciones de vigilancia ya que siempre se relacionan las actividades y las evaluaciones del riesgo o las consecuencias en el mundo real de lo que se ha inferido en el mundo simbólico de las ontologías.
- Realimentación y proceso de mejora (L4): Está dirigido a mejorar los procesos en los diferentes niveles. En esta ocasión la capa de contexto (alto nivel) envía sugerencias a la capa de seguimiento (bajo nivel). Realmente L4 consiste en la representación de estas sugerencias que han sido elaboradas entre L1 y L3.

Cada uno de esto niveles se corresponde con una ontología en el modelo de representación del conocimiento de la capa de contexto y son llamadas respectivamente: TREN, SCOB, ACTV e IMPC.

La interpretación de los datos adquiridos con el conocimiento de contexto se lleva a cabo en varios pasos:

1. Abstracción de la información de tracking: En esta fase la información proporcionada por el sistema de tracking, es transformada en instancias de ontologías, a través de la

interfaz de comunicación, esto actualiza el modelo abstracto de la escena de la capa de contexto.

2. Correspondencia entre tracks y objetos: Las representaciones simbólicas son creadas como resultado del razonamiento con las entradas provenientes de la capa de seguimiento.
3. Reconocimiento de las actividades a partir de la descripción de los objetos: Las actividades son reconocidas.
4. Evaluación de las amenazas: Las actividades son evaluadas para conocer su impacto en el sistema.
5. Cálculo de las recomendaciones por la capa de contexto: Se realiza de forma paralela a la interpretación de la escena. Las recomendaciones pueden ir desde sugerencias sin un efecto directo hasta indicaciones muy precisas.

3.3.2 EL MODELO DE CONOCIMIENTO

El modelo de la capa de contexto es una representación simbólica de la dinámica, el entorno y los comportamientos de los objetos de la escena analizada por la capa de seguimiento [1].

El modelo de la capa de contexto abarca varias ontologías según la información que se esté utilizando. Las ontologías generales, incluyen un conocimiento más abstracto, mientras que las ontologías específicas refinan los conceptos, de acuerdo al dominio tratado.

El contenido de las ontologías generales es un equilibrio entre lo general y lo útil. Deben ser generales para reutilizar ese conocimiento y poder usarlas en otros dominios, pero por otro lado, deben de incluir todos los elementos de descripción posibles.

El modelo puede incluir instancias de objetos previas al razonamiento que describen el contexto del escenario, a estos datos preliminares se les denomina información de contexto. Esta información suele utilizarse para objetos de la escena de carácter estático.

Un ejemplo del uso de esta información puede ser la instanciación de todos los objetos de tipo puerta, cuyas coordenadas se saben de antemano puesto que el escenario es ya conocido. Toda esta información facilitaría el razonamiento sobre los accesos de las pistas al entorno, si se detectase una pista que no accediese por una puerta, existiría un razonamiento asociado dependiendo del tipo de aplicación que iría desde eliminar la pista hasta dar un aviso a un operador a partir del nivel de evaluación de riesgo.

3.3.3 LAS ONTOLOGÍAS EN EL MODELO DE CONOCIMIENTO

Como se ha visto en el punto **3.3.1 La estructura de la capa de contexto** el modelo de la capa de contexto, abarca varias ontologías que representan, información de seguimiento (TREN), objetos de una escena (SCOB), actividades (ACTV), impactos (IMPC) y recomendaciones para la capa de bajo nivel.

Necesariamente la granularidad deseada y cantidad de información de contexto gestionada por el sistema, determina la capacidad de detalle en la interpretación de las escenas. En los niveles más bajos L0 y L1, no existe el análisis de las relaciones entre los objetos, sin embargo, en el nivel L2 la escena es interpretada en términos de propiedades de objetos y relaciones entre estos objetos.

El conocimiento sobre los objetos se representa en la capa de contexto, con las ontologías de nivel L1/L2 TREN (TRacking ENtities) y SCOB (Scene OBjects).

- TREN, es una representación de la información en crudo de la capa de seguimiento, no añade nueva información a la pista a parte de los valores suministrados.
- SCOB, es usada para representar objetos de la escena desde una perspectiva cognitiva. En vez de pistas sin especificar concreta el tipo de objetos que son necesarios para comprender la escena [1]. Este pequeño paso provoca en realidad un cambio de perspectiva, considerar el tipo de objeto en el que se convierte una pista también es considerar como se relaciona con el entorno, sus posibles comportamientos, sus propiedades... en general es afinar las posibles interpretaciones de la escena.

La ontología TREN

Los conceptos centrales de la ontología de seguimiento de entidades son Frame y Track.

Un Frame se identifica por un valor numérico y son señalados con una marca de tiempo [1]. Las entidades de tipo Frame pueden asociarse a sus imágenes correspondientes tal y como son capturadas por la capa de seguimiento, estas imágenes son individuos (hechos concretos de un concepto) de RawData.

Los Tracks también son identificados numéricamente. No existe una marca de tiempo para ellos puesto que perduran en el tiempo, por tanto, es necesario representar su evolución temporal. El problema de esta situación es la necesidad de asociar a cada Track varios conjuntos de valores correspondientes a sus propiedades, que son validas solo durante algunos frames. Para ello se asocian un conjunto de TrackSnapshots a cada Track. Cada TrackSnapshots que tenga propiedades inicializadas es válido durante varios fotogramas.

Las instancias de TREN son creadas por la capa de seguimiento a través de la interfaz de comunicación. Las instancias de TrackSnapshot son creadas por ejemplo cuando la capa de seguimiento detecta un cambio de posición o una oclusión. Cuando la capa de seguimiento detecta un cambio de la posición de un track, una nueva instancia de ActiveTrackSnapshot con los valores actualizados, es asociada a esa instancia de Track. El valor de la propiedad que indica cuando termina de estar activo el susodicho track (a través del valor de la propiedad isValidInEnd) debe ser modificado del frame anterior al actual. Del mismo modo, si ese track es ocluido en un momento del análisis, debe crearse una instancia de OccludedTrackSnapshot.

Sería posible en esta misma ontología hacer cálculos para propiedades complejas como la distancia o las relaciones espaciales.

La ontología SCOB

La ontología SCOB ofrece un modo de representar objetos de una escena y sus propiedades, estos objetos son entidades del mundo real con apariencia visual.

A diferencia de la ontología TREN, SCOB es utilizada para aplicaciones concretas, es por ello mucho menos general, de este modo es capaz de representar conceptos como persona, coche, perro...

SCOB se encuentra entre los niveles L1 y L2 de JDL debido a que principalmente contiene conocimiento acerca de objetos, también contiene relaciones entre objetos, considerando su interacción.

En cuanto a su implementación, el concepto principal es denominado SceneObject, es un concepto en el que se incluyen todos los objetos de interés de la escena, ya sean de tipo dinámico, como por ejemplo un track o de tipo contextual, como pueden ser una columna o una puerta.

- El primer tipo de objetos, los de tipo dinámico, pertenecen al concepto TrackedObject y son evidentemente los objetos asociados a las pistas detectadas (importadas de la ontología TREN). Los tracks son analizados para determinar de qué objeto del mundo real se puede tratar y en consecuencia se crea una instancia de tipo TrackedObject.
- El segundo tipo de objetos, los de tipo contextual, pertenecen al concepto StaticObject, son los objetos cuyas propiedades no cambian con el paso de los fotogramas. Estos objetos, no son detectables por el análisis de la capa de seguimiento y tienen que ser introducidos a través de una interfaz como información del contexto para un entorno concreto.

Las propiedades de los objetos de una escena, cambian a lo largo de la secuencia, ya sea la posición, la iluminación, el comportamiento... Como es lógico, alguno de los valores de estas propiedades provienen de la ontología TREN.

Gracias a un buen diseño, es posible describir los cambios en los valores de las propiedades de cada objeto durante toda la secuencia y ampliar estas propiedades en el futuro sin sufrir dificultades.

La ontología ACTV

En cuanto a las ontologías relativas a las actividades (ACTV), proporcionan un vocabulario para describir escenas [1]. Las escenas están definidas a partir de las relaciones existentes entre los objetos que se encuentran en ellas.

Del mismo modo que en el caso de SCOB de nivel L1/L2, debían importarse instancias de TREN, en este caso, el nivel inferior (SCOB) es el que proporciona la información a la ontología de las actividades de nivel L2.

Solo las actividades muy generales, pueden ser incluidas en estas ontologías, el desarrollo de actividades de un dominio específico debe conseguirse a partir del refinamiento y especificación de estas ontologías.

La ontología IMPCT

La ontología IMPCT, del nivel L3, contiene un vocabulario para asociar un valor de evaluación a las instancias de la ontología ACTV que representan situaciones del mundo real. El valor puede consistir en un porcentaje o un número cuyo objetivo es representar el riesgo que esa actividad tiene en ese dominio. Un ejemplo simple de esto, sería un sistema de seguridad por videovigilancia, que se detectase un objeto (track de TREN) entrando en la escena por un lugar que no fuera un acceso permitido de tipo puerta (objeto de SCOB). El sistema debería dar un valor de riesgo alto, lo que provocaría una reacción que probablemente en este caso sería el encendido de una alarma.

En este ejemplo, no solo se observa la importancia de esta ontología, sino que también puede entenderse como un ejemplo de cómo la información fluye a través de los niveles del sistema de contexto. Los elementos detectados y configurados en los estamentos más bajos suben por las ontologías hasta convertirse en elementos del mundo real detectados automáticamente y cuyas acciones causan consecuencias directas en el sistema.

La ontología RECO

Es considerada como una ontología de nivel L4 debido a que está destinada mejorar la calidad de los datos adquiridos y el rendimiento. RECO incluye los conceptos y relaciones que representan acciones de la capa de seguimiento ya sean realizadas por ella misma o sugeridas por la capa de contexto.

El concepto principal de RECO es Action que incluye tanto las acciones sugeridas por la capa de contexto (SuggestedActions) como las acciones ejecutadas por la propia capa de seguimiento (PerformedActions), estas últimas instanciadas a partir de las llamadas de actualización, creación y eliminación de los tracks.

Las recomendaciones deben de estar descritas a nivel de track en la capa de contexto ya que deben ser traducidas a acciones concretas en la capa de seguimiento.

3.4 EL DESARROLLO ACTUAL

En el ámbito práctico, la capa de contexto ha sido implementada como una librería Java, que es invocada desde la capa de seguimiento (codificada en C++), a través de JNI (Java Native Interface). Esta librería ofrece métodos públicos que permiten construir los modelos de escena a través de la actualización de la representación.

La librería también contiene métodos privados para gestionar las ontologías del modelo de la capa de contexto, basados en la API de OWL y comunicados con el motor de inferencia que en la implementación actual es el razonador RACER [2].

Antes de comenzar la cadena de procesado de la secuencia de vídeo la capa de contexto ha de estar ya inicializada. Esto requiere cargar el modelo de la capa de contexto en el motor de razonamiento, lo que incluye las ontologías generales y específicas, así como las reglas de inferencia.

Actualmente la información de contexto, está incrustada en las ontologías, es decir, forma parte de su definición, lo que significa que para cada escenario es necesario cambiar la información de contexto del sistema que en todo caso afecta en su gran mayoría a la ontología SCOB. Se estudia para futuras versiones separar esta información insertándola a partir de la interfaz de comunicación en un archivo, lo que independizaría la información y la haría fácilmente modificable.

En el proceso actual de análisis de vídeo cuando la información de una pista cambia en un frame, la capa de seguimiento, realiza llamadas a la interfaz pública para actualizar el modelo de la capa contexto. Los métodos de la interfaz son capaces de transformar la información de la capa de seguimiento en instancias de ontologías de tipo TREN.

Para este desarrollo en concreto, se han implementado y probado los niveles de ontologías L1, lo que incluye la ontología TREN. Existe una representación simbólica de los elementos de cada escena, preparado para pasar al nivel de la ontología SCOB donde las pistas se corresponden con objetos del mundo real.

En el nivel de SCOB se llevan a cabo inferencias que generan nueva información, esto implica un desarrollo en el lenguaje de consultas nRQL cuyas reglas sean usadas junto con las ontologías y el motor de inferencia para este cometido.

El desarrollo de las ontologías ACTV e IMPC están todavía en fases iniciales, por la importancia del reconocimiento de actividades en un sistema de seguimiento, en el ámbito de la investigación, ACTV se llevara a cabo en un corto espacio de tiempo. El desarrollo de IMPC será un proceso dependiente del interés de las organizaciones o empresas interesadas en este tipo de tecnología, debido a que su uso tiene un objetivo más específico y un enfoque mucho más comercial que el resto de ontologías del sistema.

Una vez finalizada la ontología RECO, se conseguiría la traducción de la información simbólica elaborada por las ontologías, en acciones concretas en la capa de seguimiento lo que provocaría una relación sinérgica entre las capas, la mejora de la interpretación de una sería también una mejora en la interpretación de la información de la otra capa.

4 CADENA DE VÍDEO

En este capítulo se trata la estructura, funcionamiento y características de tracker o cadena de vídeo de la aplicación. Es importante conocer esta capa de la aplicación en detalle, puesto que, consistió en el proyecto inicial y para mejorar su rendimiento y funcionalidad se han desarrollado el resto de productos.

En primera instancia se realiza una introducción teórica de los conceptos principales de este producto, en segundo lugar se trata la tecnología OpenCV y por último, se profundiza en la estructura del desarrollo particular de este software.

4.1 INTRODUCCIÓN TEÓRICA

El planteamiento inicial del desarrollo de la cadena de vídeo, consistía en lograr una aplicación de seguimiento muy eficiente, para el procesado de vídeo en tiempo real.

El objetivo principal, era centrase sobre todo en la detección de los objetos de la escena y en el cálculo de trayectorias (seguimiento) de los objetos móviles, a lo largo del tiempo, mediante la localización de su posición en cada instante [5]. Como resultado, el software tiene por objeto

la segmentación de los fotogramas de vídeo, distinguiendo los objetos en movimiento, del fondo y etiquetándolos de forma consistente.

Para lograr sus metas, este sistema de seguimiento, tomó como esqueleto un desarrollo previo de la empresa Intel que hacía hincapié en la eficiencia y optimización del código para poder realizar análisis de vídeo en tiempo real. Esta estructura utiliza como lenguaje de programación C++ y dentro de su código se utilizan funciones de las librerías OpenCV, diseñadas específicamente para desarrollos en de aplicaciones de visión por computador.

La arquitectura de esta aplicación se basó en una cadena de vídeo, con diferentes módulos que se ejecutaban de forma secuencial y que se correspondían con las sucesivas fases del proceso de seguimiento, cuyos algoritmos, pertenecientes a cada módulo, eran intercambiables en cada ejecución. Todos los pasos de la secuencia de vídeo de entrada, se llevaban a cabo para cada fotograma.

En la actualidad, el tracker mantiene esta estructura y los módulos principales del en su implementación son:

- Detección de movimiento.
- Asociación entre detecciones o blobs y pistas o tracks. Lo que incluye; predicción, asignación y actualización.
- Creación y eliminación de tracks.
- Generación de trayectorias.

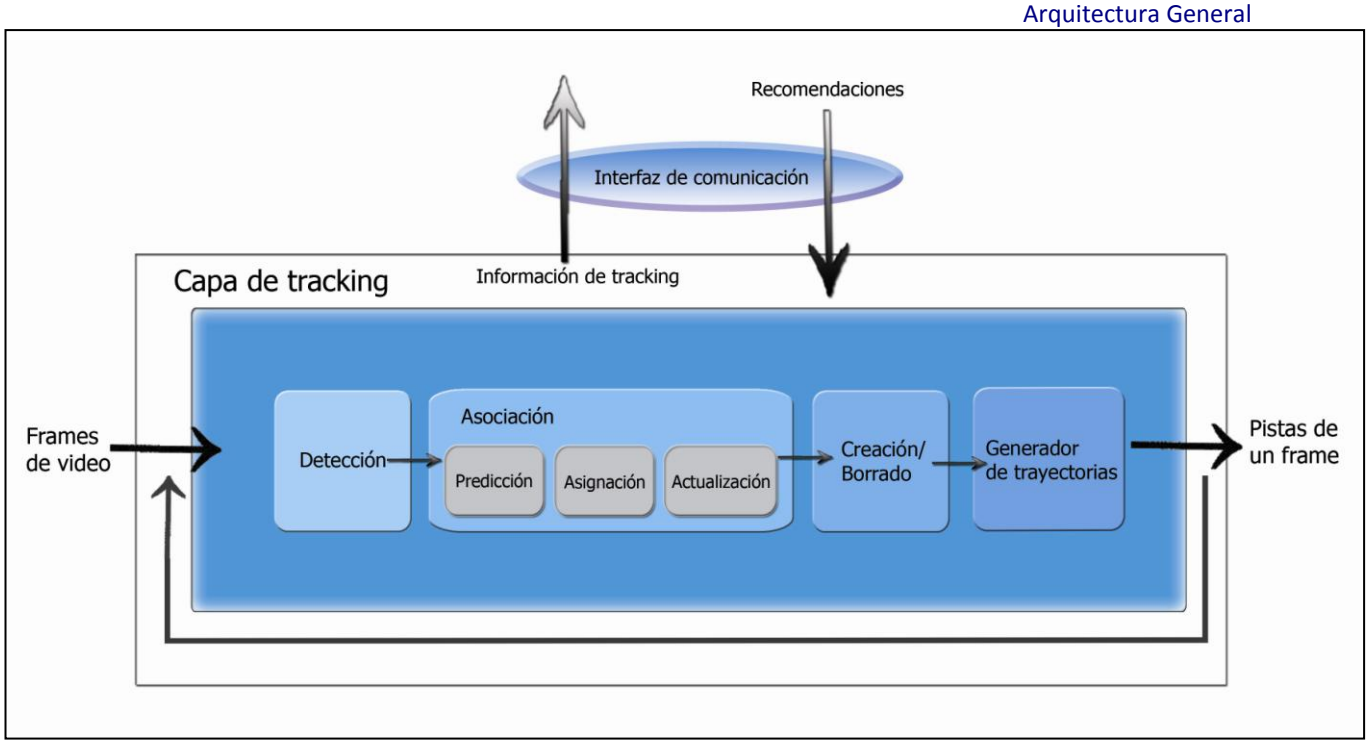

Ilustración 3

La entrada del analizador, es un vídeo en formato .avi sobre el que la aplicación realiza el análisis de visión. La salida de cada módulo se convierte en la entrada al módulo siguiente, de ahí, su condición de secuencial. Los cambios en la detección de las pistas y el cálculo de trayectorias de los objetos de la escena, pueden visualizarse en una ventana de reproducción de vídeo que representa la salida definitiva del tracker.

Existen ciertos datos que tras el análisis son realimentados debido a que la información sobre frames anteriores puede condicionar el análisis de los siguientes fotogramas.

A posteriori, este sistema de tracking, se ha convertido en una capa de bajo nivel, de un sistema más grande, llamada capa de seguimiento; un sistema que además de utilizar estos algoritmos de seguimiento, se ayuda de una capa de alto nivel.

Esta capa de alto nivel, es la denominada, capa de contexto, capaz de obtener los datos de la capa inferior a través de la interfaz de comunicación que hace a las capas independientes, crear un modelo simbólico a partir de estos datos, razonar y obtener nueva información oculta para la capa inferior y enviar esta capa, recomendaciones para mejorar el análisis, según indique el razonamiento anteriormente citado.

4.2 LA CAPA DE SEGUIMIENTO O TRACKER

El tracker cuya arquitectura se ha descrito en **4.1 Introducción teórica** queda enmarcado como la capa de bajo nivel del sistema de seguimiento.

4.2.1 LOS MÓDULOS DEL TRACKER

Como se ha dicho anteriormente, dentro de la capa de bajo nivel existen una serie de módulos que se reparten el procesamiento del análisis dependiendo del tipo de tarea que se lleve a cabo. En esta imagen puede observarse el bucle de procesado resultante de la interacción entre los módulos.

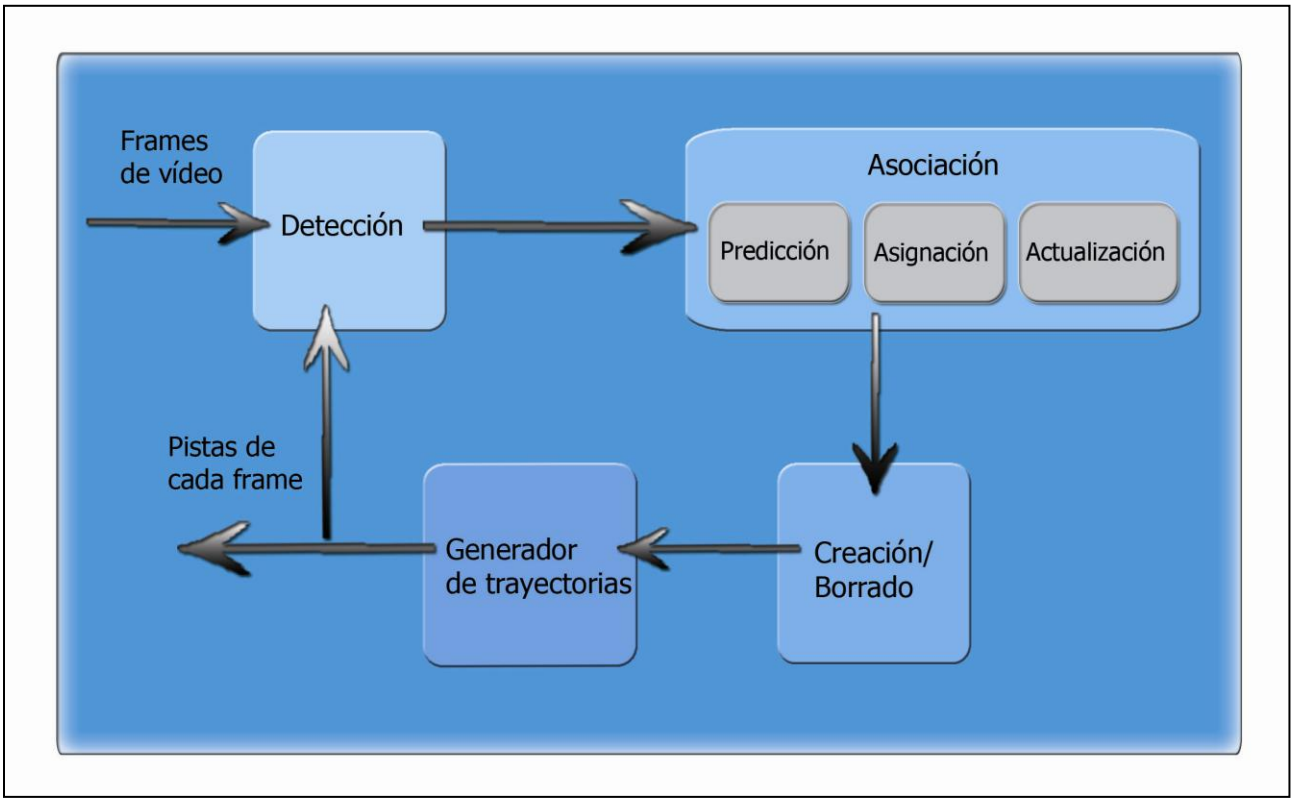

Ilustración 4

Detector de movimiento:

Cuando se realiza un sistema de seguimiento de pistas para vídeo en tiempo real, es necesario abordar en primer lugar el problema de la detección de las pistas que se desean seguir.

Traducido a los sistemas de vídeo esto significa, determinar que pixeles de la imagen pertenecen a objetos en movimiento y cuales al fondo [5]. En este contexto el fondo son, tanto los pixeles de los objetos que se encuentran de forma estática en la escena, por ejemplo, el suelo, las paredes, los muebles, etc., como también aquellos que presentan movimiento pero no son objetos en los que el seguimiento deba mostrar interés, ejemplos de estos son las hojas de un árbol en el viento, el movimiento de las olas en el mar, etc.

Los elementos del primer grupo, pueden descartarse debido a que los pixeles que pertenecen a estos objetos no se mueven y el sistema de seguimiento puede centrarse solo en los que se

mueven. Muchos de los elementos de este segundo grupo pueden descartarse por un tamaño reducido (las hojas un árbol) o por la intensidad o color que emitan.

Todas estas cuestiones se incluyen en el modelo del fondo, gracias al cual, puede decidirse si un píxel pertenece a este o a un objeto de interés.

Existen 3 algoritmos implementados para intercambiar en este módulo:

- FG_0: Consiste en la detección de objetos en primer plano, en videos con fondos de alta complejidad.
- FG_OS: Es una versión simplificada de FG_0.
- FG_1: Se basa en modelos adaptativos para el seguimiento en tiempo real.

Asociación entre blobs y tracks:

Una vez se ha obtenido la máscara de movimiento, se extraen las características para actualizar la posición de los objetivos reconocidos en cada instante, así como su tamaño.

Para realizar la asociación, se puede predecir, por ejemplo con un filtro de Kalman, la posición de cada objetivo en cada instante y asignar cada blob a su objetivo más cercano en base a alguna distancia definida.

Existen muchas estrategias definidas en el sistema en forma de algoritmos para este módulo, algunos de los más interesantes son [5]:

- TABU: Una búsqueda TABU determina que blobs pertenecen a cada campo.
- MSPF y CCMSPF: Son 2 variantes de seguimiento utilizando filtros de partículas.
- EDA, UMDA, CGA, GA y PBIL: Realizan el seguimiento utilizando algoritmos evolutivos de asociación.
- Rules: Realiza el seguimiento utilizando un conjunto de reglas heurísticas para realizar la asociación.
- EM: Realiza el seguimiento utilizando para la asociación agrupamiento EM.
- Otros: MS, MSFG, CC, Kalman.

Creación y eliminación de objetivos:

Los objetivos detectados en una escena, pueden desaparecer durante un determinado número de imágenes, si esto ocurre el sistema de seguimiento debe eliminar esas pistas. Existen una serie de reglas específicas para eliminar los objetivos del sistema. El algoritmo que las

implementa establece una puntuación, y cuando dicha puntuación supera un umbral, el blanco debe desaparecer.

Las reglas para establecer dicha la puntuación son [5]:

- Si el ancho y el alto, son mayores de 4 y el centro del blanco se encuentra dentro de la imagen:
    - Si menos del 10% de la superficie del blanco contiene píxeles en movimiento, la puntuación sumara 1.
    - Si no la puntuación será 0.
- Si no sumara 1.

También es posible que nuevos objetivos aparezcan en la escena, en ese caso sería necesario inicializarlo a partir de la máscara de movimiento proporcionada por el módulo de detección [5].

Actualmente existen en el tracker 2 estrategias de inicialización similares, ambas se basan en la realización de un seguimiento preliminar en aquellas zonas de la máscara de movimiento que no han sido asociadas con ninguna pista. Estas 2 estrategias están denotadas en la aplicación como:

- BD_CC.
- BD_Simple, cuya diferencia con la anterior es que detecta los objetivos por el movimiento uniforme de componentes conectados entre sí en la máscara de movimiento.

Generación de trayectorias:

Este módulo tiene como cometido almacenar las trayectorias de los objetivos durante el seguimiento.

Existen en el tracker 2 algoritmos distintos para el almacenado de estos datos [5]:

- YML: Almacena las posiciones de los blancos, mediante la serialización de sus posiciones utilizando el formato YAML (http://www.yaml.org). La funcionalidad para realizar dicho almacenaje, la proporciona OpenCV.
- En bruto: Almacena las posiciones de cada blanco en cada instante de tiempo separada por una coma.

4.2.2 LOS MÓDULOS ADICIONALES

A efectos teóricos puede explicarse la arquitectura básica con la que se trabaja la capa de seguimiento a partir de los 4 módulos vistos anteriormente. En todo caso esta arquitectura solo se considera la base de la capa de seguimiento, cuya implementación consta de módulos adicionales.

Postprocesado de trayectorias:

Una vez se ha llevado a cabo la actualización de los objetivos, es posible realizar una corrección de sus valores para suavizar el efecto causado por los errores, esto se lleva a cabo en la fase de Postprocesado de trayectorias, cuyo único algoritmo implementado actualmente en el tracker es el filtro de Kalman.

Análisis de trayectorias:

El estudio de trayectorias y la predicción de su comportamiento corren a costa de este módulo.

Existen multitud de algoritmos implementados para los diferentes tipos de análisis:

- HistPVS: Histograma de cinco características.
    - Posición del objetivo en el eje x.
    - Posición del objetivo en el eje y.
    - Velocidad del objetivo en la componente x.
    - Velocidad del objetivo en la componente y.
    - Estado del objetivo.
- HistP: Histograma de dos características.
    - Posición del objetivo en el eje x.
    - Posición del objetivo en el eje y.
- HistPV: Histograma de cuatro características.
    - Posición del objetivo en el eje x.
    - Posición del objetivo en el eje y.
    - Velocidad del objetivo en la componente x.
    - Velocidad del objetivo en la componente y.
- HistSS: Histograma de dos características.

- Posición inicial.
- Posición final.

- TrackDist: Algoritmo que compara los objetivos con trayectorias almacenadas en un fichero.
- IOR: Integra varios análisis mediante el uso de la operación OR.

Módulo de control:

El módulo de control actúa como pegamento entre todos los módulos de la capa de seguimiento, toma el control tras finalizar el procesado en cada fase y se encarga de inicializar cada uno de los algoritmos pertenecientes a los módulos del sistema de seguimiento.

En este diagrama se observa de modo gráfico la secuencia de procesamiento completa implementada en el sistema:

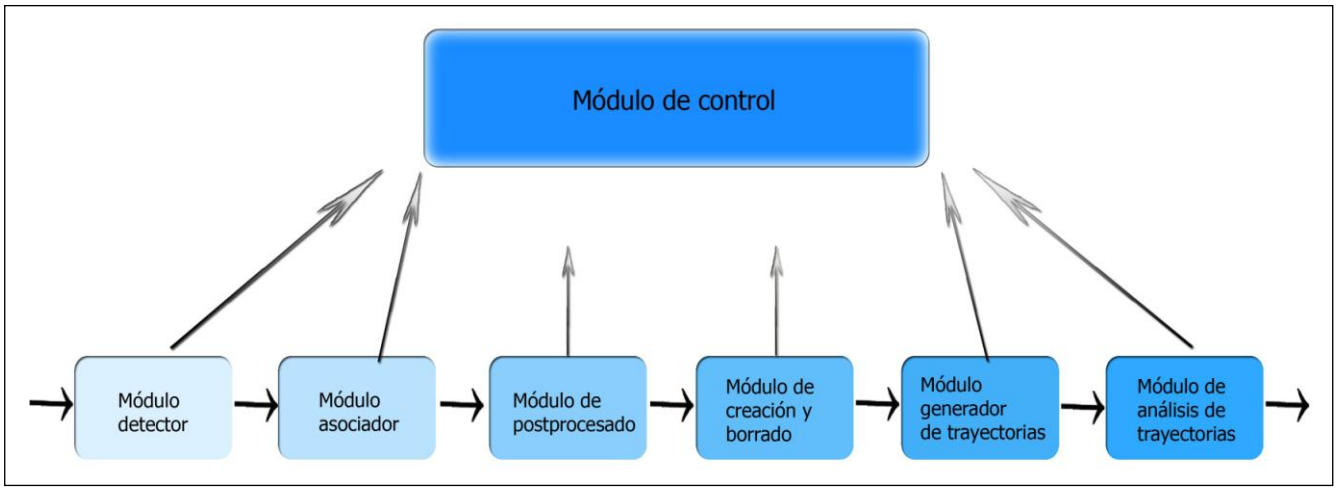

Ilustración 5

4.2.3 LA INTERFAZ DE COMUNICACIÓN EN EL TRACKER

Debido a su independencia, es necesario que exista una comunicación entre las capas a través de una interfaz, de este modo es posible transmitir los datos entre la capa de razonamiento de bajo nivel y la capa de razonamiento de alto nivel.

Desde un punto de vista abstracto, la interfaz de comunicación, se toma como un elemento independiente, escrito en código no nativo, que hace enlace entre 2 vías de razonamiento. En la práctica, esta interfaz, es albergada por el tracker hasta el punto de que el mismo decide cuando se inicializa y cuando se realizan las llamadas. Los objetos que forman la interfaz son instanciados desde el tracker, por tanto está completamente incrustada en los módulos del tracker.

Durante el proceso de tracking, la interfaz de comunicación, es invocada cada vez que la información de una pista cambia, lo que actualmente ocurre en las fases de actualización, inicialización y borrado.

- La actualización envía los datos característicos del track, su identificador, el frame en el que se realiza la operación de actualización, la posición en 2 dimensiones, la anchura y altura.
- La creación y borrado envían a la capa de contexto la información de identificación del track y del frame en el que es creado o borrado.

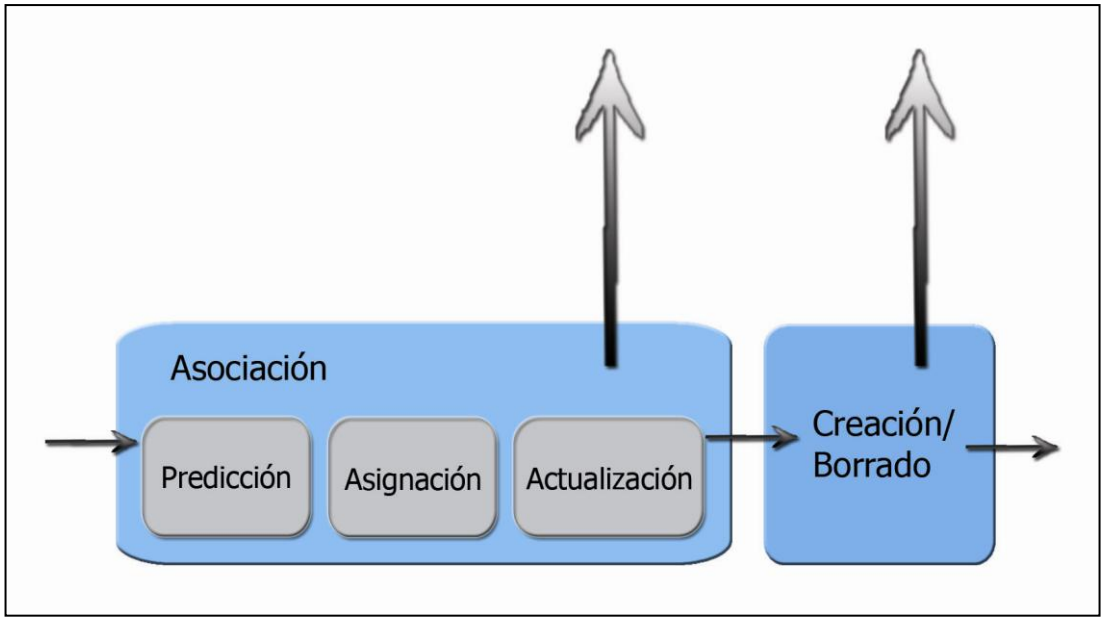

Ilustración 6

Todas estas peticiones son realizadas a través de llamadas a la interfaz de comunicación, que renuevan el modelo simbólico de la capa de contexto de acuerdo con las percepciones halladas en la capa de tracking. Las percepciones o información de los tracks, se pasa a la interfaz como datos de tipo numérico, esta los transforma en instancias de ontologías que a su vez provocan la modificación del modelo de la capa de contexto. Estas actualizaciones del modelo de contexto hacen que se disparen procesos de razonamiento y estos procesos de razonamiento resultan en recomendaciones que cruzan de nuevo el camino entre las 2 capas, modificando el comportamiento de la capa de seguimiento o tracker.

Es necesario, expresar las recomendaciones de tal forma que la capa de seguimiento pueda convertirlas en operaciones concretas.

4.3 OPENCV

OpenCV es un conjunto de librerías de código abierto, para el desarrollo de aplicaciones que tienen que ver con visión por computador. Estas librerías, están escritas en C y C++ y corren bajo Linux, Windows y Mac OS X [4].

OpenCV fue diseñado para ofrecer una gran eficiencia computacional, de modo que los sistemas creados a partir de estas librerías pudieran enfocarse a aplicaciones de tiempo real. De hecho, estas librerías se escribieron en un lenguaje optimizado, que forma parte del lenguaje C y que puede utilizar la tecnología de procesado en varios núcleos.

El principal objetivo de OpenCV es facilitar una infraestructura simple de visión por computador, que ayude a la gente a construir aplicaciones de visión sofisticadas rápidamente. Las librerías contienen más de 500 funciones que abarcan capacidades desde la inspección de productos de una fábrica, calibrado de cámaras y seguridad, hasta visión estereoscópica e incluso aplicaciones para robots.

Las funciones más comunes, relativas a la resolución de los problemas de visión artificial, provistas por estas bibliotecas pueden clasificarse en sus distintos paquetes:

- CXCORE:
    - Proporciona estructuras básicas tales como puntos o áreas útiles para los algoritmos.
    - Puede realizar manipulación de vectores y matrices, que normalmente representan conjuntos de píxeles pertenecientes a una imagen, con operaciones algebraicas.
    - Aporta estructuras dinámicas, utilizadas para gestionar de forma correcta la memoria y las secuencias, tales como colas, listas, arboles...
    - Soporta funciones de dibujado de curvas y figuras, esto es muy útil a la hora de mostrar por pantalla las detecciones de elementos.
    - Es capaz de cargar y guardar datos, con las funciones de persistencia.
    - Contiene una gestión propia de los errores.
- CvReference ofrece:
    - Procesado y análisis de imágenes, lo que incluye inicializar y eliminar imágenes en memoria y modificar el valor de los píxeles en diferentes espacios de color.
    - Análisis estructurales.

  - Análisis y seguimiento de objetos de referencia en movimiento.
  - Algoritmos de reconocimiento de patrones
  - Técnicas estereoscópicas para la reconstrucción 3D y es capaz de realizar el calibrado de cámaras.

- CvAux aporta:
  - Funciones de correspondencia estereoscópica.
  - Funciones de 'view morphing'
  - Funciones de seguimiento en 3D.
  - Funciones con autovectores.
  - Funciones embedidas de modelos ocultos de Marcov.

- HighGUI proporciona:
  - Funciones simples para las interfaces graficas de usuario, tales como, poner nombre a una ventana o modificar su tamaño.
  - Cargar y salvar imágenes.
  - Funciones de entrada y salida de imágenes y vídeo, como la captura de imágenes desde una cámara o la escritura de imágenes en ficheros.

Debido a que la visión y el aprendizaje por computador están muy unidos, OpenCV incluye una completa librería de propósito general, llamada Machine Learning Library (MLL). Esta sublibrería, se centra en patrones estadísticos de reconocimiento y agrupación y es muy útil para las tareas de visión.

Algunas de sus principales funciones son:

- Clasificadores bayesianos.
- Algoritmos de clasificación por vecindad.
- Maquinas de soporte vectorial.
- Árboles de clasificación.
- Redes de neuronas.
- Boosting.
- Random trees

- Expectation-Maximization (EM)

Existen también rutinas de muchas de las áreas en las que se investigan algoritmos relativos a la visión por computador, que han sido optimizadas desde los niveles más bajos de abstracción. Todas estas rutinas se agrupan en librerías IPP (Integrated Performance Primitives) y pueden ser compradas a la compañía Intel. OpenCV, es capaz de, automáticamente, utiliza la librería IPP apropiada en tiempo de ejecución, una vez instaladas.

5 INTEGRACION CADENA DE VÍDEO - ONTOLOGIAS

Este capítulo trata de explicar cómo se ha llevado a cabo el puente entre la capa de seguimiento y la capa de contexto a través de la interfaz de comunicación. En primera instancia se hace una introducción de las tecnologías usadas para llevar a cabo este enlace, posteriormente se explica pormenorizadamente la interfaz de comunicación utilizada para unir las capas y para terminar se expone detalladamente el wrapper desarrollado específicamente para este caso particular.

5.1 INTRODUCCIÓN

Durante la construcción de la aplicación de análisis de vídeo que se presenta en este proyecto, fue necesario, el desarrollo de un wrapper, para poder unir las capas de contexto y seguimiento, escritas en lenguajes distintos.

5.1.1 ¿QUÉ ES UN WRAPPER?

Un wrapper, es un adaptador, que tiene la capacidad de convertir una interfaz de una clase incompatible con la especificación usada, en una interfaz compatible, permitiendo el acceso al código nativo de otro lenguaje, usando así sus funciones o bibliotecas. Es posible utilizar la interfaz original con la que se deseaba trabajar, traduciendo las llamadas a partir de una interfaz propia.

Estos adaptadores, también son responsables de transformar los datos utilizados en las llamadas, para que tengan un formato adecuado. El ejemplo más simple, sería la transformación los datos booleanos en C que utilizan los valores 0 y 1 de un número entero, en los valores ‘true’ y ‘false’ del tipo de datos correspondiente en Java.

5.1.2 JAVA NATIVE INTERFACE (JNI)

Los diseñadores de Java a principios de 1997, crearon un mecanismo para conseguir que una aplicación Java llamara a métodos de C/C++. Este mecanismo utiliza una colección de APIs de C++ conocidos como Java Native Interface [3]. Esto permite realizar llamadas desde Java a métodos nativos de C o de C++. De hecho toda la librería de Java, desde la lectura de ficheros hasta el soporte de red, en realidad están escritas en C y C++ y son llamados utilizando estos métodos nativos. Por tanto JNI fue una herramienta creada, en principio, por y para los propios desarrolladores de Java.

Ocurre que existe también la necesidad en algunos casos de realizar un código en C o C++ que base su implementación en métodos Java. JNI también permite desde estos lenguajes, llamar a métodos Java. Esta interfaz de doble dirección puede resumirse en el siguiente diagrama:

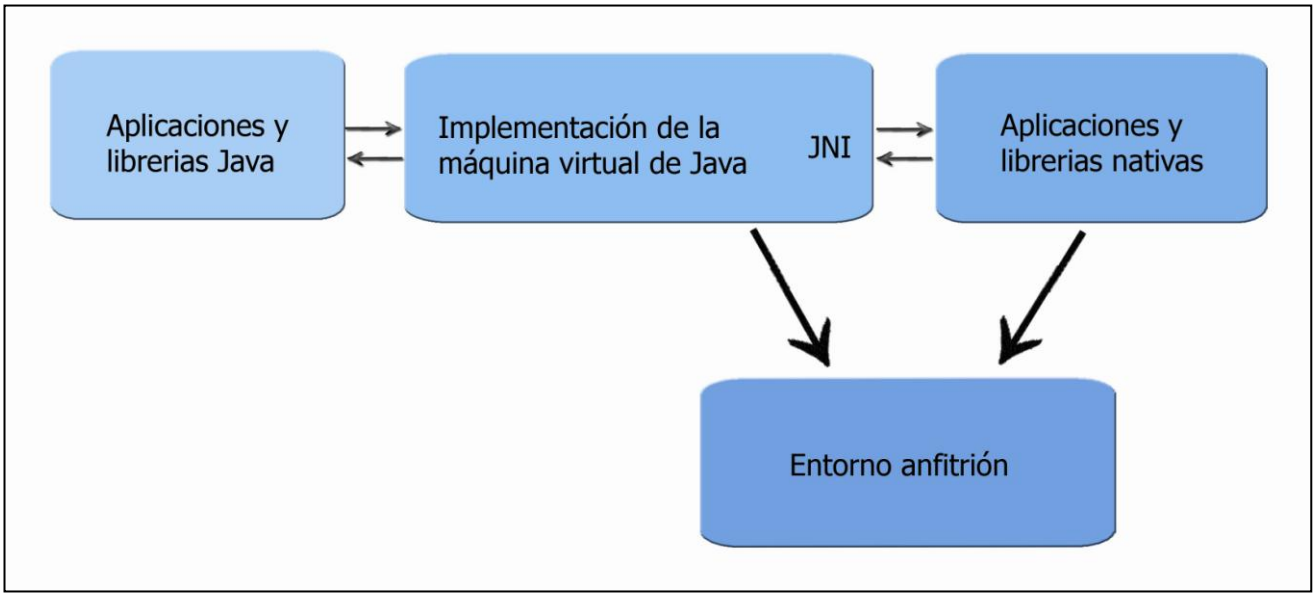

Ilustración 7

JNI soporta una interfaz de invocación que permite incrustar una implementación de la máquina virtual en aplicaciones nativas. El proceso consiste en que la aplicación nativa puede enlazar con una librería que implementa la máquina virtual de Java y a partir de entonces usar la interfaz de invocación para ejecutar componentes software escritos en el lenguaje de programación Java

Esta capacidad es la que se aprovecha en la implementación de este wrapper, dándole la vuelta a la utilidad inicial de JNI, llamando así a aplicaciones Java (capa de contexto), desde una aplicación en C++ (capa de seguimiento).

El uso de la Interfaz Nativa de Java (JNI) en este contexto queda justificado, ante la necesidad ineludible de interoperar con un código distinto al que se propuso al inicio del proyecto.

Evidentemente, se observan ciertos riesgos cuando se hace uso de esta utilidad:

- Las aplicaciones Java que utilizan JNI no suelen estar siempre preparadas para utilizar diferentes entornos. La parte de la aplicación escrita en Java, es portable, pero es necesario hacer una recopilación de la parte escrita en código nativo.
- Es necesario tener un cuidado extraordinario al escribir aplicaciones que utilizan JNI, debido a que un malfuncionamiento de un método nativo puede corromper la aplicación al completo.

No hay que olvidar que existen otras utilidades para facilitar la interacción entre Java y C++, que aunque no se han utilizado en este proyecto, podrían ser interesantes en un futuro. Una de ellas es Jace un conjunto de librerías y programas que ayudan a escribir código en C++ que integra llamadas a métodos implementados en la máquina virtual de Java [3].

5.1.3 FUNCIONES JNI

En este punto se definen y describen las funciones JNI utilizadas para la construcción del wrapper, como introducción a lo que se tratara en los puntos posteriores.

Las funciones JNI, se enmarcan en 4 categorías dependiendo del lugar en el que hayan sido definidas y de cuál es su cometido.

- La primera categoría, son un conjunto de funciones nativas, exportadas por la implementación de la máquina virtual. Estas funciones son parte de la interfaz de invocación y pueden ser usadas para diversas tareas, por ejemplo, la creación de una instancia de la máquina virtual en una aplicación nativa.
- La segunda categoría, es la interfaz JavaVM, que representa una instancia de la máquina virtual. Esta interfaz, proporciona funciones que permiten, por ejemplo, subprocesos o hilos nativos en la instancia de la máquina virtual.
- En tercer lugar, si existe una librería nativa que implementa métodos nativos, pueden exportarse funciones, que son llamadas, cuando una implementación de la máquina virtual carga y descarga la librería nativa.
- Por último, la interfaz JNIEnv da soporte a las características de JNI, tales como, la creación de objetos, las llamadas a los métodos, el acceso a los campos…

Categoría 1: Funciones directamente exportadas de la interfaz de invocación.

Una implementación de la máquina virtual, directamente exporta como parte de la interfaz de invocación, las 3 funciones siguientes:

- `JNI_GetDefaultJavaVMInitArgs`
- `JNI_CreateJavaVM`
- `JNI_GetCreatedJavaVMs`

La función `JNI_GetDefaultJavaVMInitArgs`, proporciona los valores de los argumentos de inicialización por defecto usados para crear la instancia de la máquina virtual,

esta información es específica de la versión 1.1 de JDK y aunque en las versiones posteriores no se ha utilizado, se mantiene para conseguir compatibilidad hacia atrás.

La función `JNI_CreateJavaVM` crea una instancia de la máquina virtual, de acuerdo con un conjunto de argumentos de inicialización.

La función `JNI_GetCreatedJavaVMs` devuelve todas las instancias de maquinas virtuales, que hayan sido creadas en un proceso. Una implementación normal de JNI, no necesita crear más de una máquina virtual en el mismo proceso. De hecho, ninguna versión de JDK lo permite.

Categoría 2: La interfaz JavaVM

La interfaz JavaVM, es un puntero a un puntero a una tabla de funciones, las 4 entradas útiles en la actualidad (el resto están reservadas para ser compatibles con la interfaz COM de Microsoft), son parte de la interfaz de invocación:

- `DestroyJavaVM`
- `AttachCurrentThread`
- `DettachCurrentThread`
- `GetEnv`

Un puntero a la interfaz JavaVM, representa una instancia completa de una máquina virtual, y es válido para todos los hilos que esta contenga.

La función `DestroyJavaVM` desmonta la instancia de la máquina virtual, denotada por el puntero de la interfaz JavaVM.

La función `AttachCurrentThread` establece el hilo nativo actual, para ejecutarse como parte de una instancia de máquina virtual. Una vez el hilo se adjunta a la instancia de la máquina virtual, entonces, se pueden hacer llamadas a las funciones JNI, para realizar tareas como el acceso a objetos y la invocación de métodos.

La función `DettachCurrentThread` informa a la instancia de la máquina virtual, que el hilo actual, ya no necesitar emitir llamadas a funciones JNI, esto permite a la implementación máquina virtual llevar a cabo limpiezas y liberar recursos.

La función `GetEnv` tiene 2 propósitos, por un lado, puede ser usado para comprobar si el hilo actual, pertenece a la instancia de la máquina virtual, por otro lado, puede ser usado para obtener otras interfaces como la interfaz JNIEnv de un puntero que contenga la interfaz JavaVM.

Categoría 3: Funciones definidas en librerías nativas

A partir de la versión 1.2 de JDK se permite exportar funciones adicionales para ser invocadas cuando una implementación de la máquina virtual carga y descarga librerías nativas.

Cuando se carga una librería nativa, se busca y se invoca la entrada de la función de exportación `JNI_OnLoad`.

Cuando se descarga una librería nativa, se busca y se invoca la entrada de la función de exportación `JNI_OnUnload`.

Categoría 4: La interfaz JNIEnv

La interfaz JNIEnv soporta el núcleo de las características de JNI. Todas las implementaciones de máquina virtual, pasan como argumento, un puntero a una interfaz JNIEnv en cada método nativo. Este puntero, es válido solo para un hilo en particular, no se extiende al resto de hilos ejecutados en la instancia de máquina virtual. JNIEnv, es un puntero a un puntero a una tabla de funciones.

5.1.4 LA INTERFAZ JNIENV

Como introducción al funcionamiento del wrapper se analiza la interfaz JNIEnv. Este punto cubrirá en detalle todas las entradas importantes de la interfaz de JNIEnv utilizadas en la implementación. Existen diferentes categorías según el tipo de operación que se ejecuta y no se debe olvidar que JNI es una herramienta para realizar llamadas desde un lenguaje a otro, por lo tanto lo que se expone aquí son las llamadas utilizadas como puente entre estos lenguajes.

Operaciones de interfaz y clase

FindClass: Devuelve una referencia de una interfaz o clase del tipo pedido según el nombre introducido como argumento. Esto se consigue gracias a que localiza el cargador de clases asociado con el método nativo.

La signatura de este método es:

```
jclass FindClass (const char* name);
```

`name`: Es el descriptor de la clase o la interfaz que el método debe devolver.

Operaciones con objetos

NewObject: Asigna un objeto y ejecuta uno de sus constructores. Devuelve una referencia local a un objeto o un valor nulo si el objeto no pudo construirse.

El identificador del método constructor, se consigue llamando al método GetMethodID con "<init>" como nombre del método y "V" (void) como tipo de datos de retorno. El constructor, debe estar definido en la clase apuntada por el argumento `clazz`, no en una de sus superclases, además `clazz` no debe referirse a un array. Todos los argumentos que se pasan posteriores a `methodID` son pasados al constructor que se va a invocar.

La signatura de este método es:

```
jobject NewObject (jclass clazz, jmethod methodID, …);
```

`clazz`: Referencia a una clase cuya instancia va a ser creada.

`methodID`: El identificador del método constructor, que sera ejecutada al crear la nueva instancia.

Argumentos adicionales: Argumentos que se pasan al constructor de la clase.

Llamadas a métodos de una instancia

GetMethodId: Este método devuelve el identificador del método buscado o en caso de que falle NULL. El método es determinado por su nombre y por su descriptor. En el caso de los constructores se utiliza "<init>" como nombre y "V" (void) como tipo de retorno.

La signatura de este método es:

```
jmethodID GetMethodID(jclass clazz, const char *name, const char *sig);
```

`clazz`: Es la referencia a la clase o interfaz a la que pertenece el método.
`name`: El nombre del método.
`sig`: El descriptor del método. Contiene una nomenclatura específica explicada en el apartado **5.1.5 El tipado de datos en JNI**.

Familia Call<Type>Method: Esta familia de funciones contiene 10 métodos. Estos invocan los métodos de instancias, pertenecientes a otro lenguaje, en este caso Java, utilizando un identificador. Todos los argumentos del método que se está tratando deben ponerse después del identificador del método.

Call<Type>Method	<Native Type>
CallVoidMethod	void
CallObjectMethod	jobject

CallBooleanMethod	jboolean
CallByteMethod	jbyte
CallCharMethod	jchar
CallShortMethod	jshort
CallIntMethod	jint
CallLongMethod	jlong
CallFloatMethod	jfloat
CallDoubleMethod	jdouble

La signatura general de estos métodos es:

```
<NativeType> Call<Type>Method (jobject obj, jmethodID methodID, …);
```

`obj`: Es la referencia al objeto de cual se invocará el método.

`methodID`: Denota el identificador del método para ser invocado.

Argumentos adicionales: Argumentos que se pasan al método llamado.

Llamadas a métodos estáticos

GetStaticMethodId: Realiza una búsqueda simbólica de una clase y devuelve el identificador de un método estático de la clase denotada por `clazz` o un valor nulo si la operación falla. El método debe estar definido en la clase pasada por argumento o en una de sus superclases.

La signatura de este método es:

```
jmethodID GetStaticMethodID (jclass clazz, const char *name, const char* sig);
```

clazz: Referencia al objeto de la clase cuyo método estático va a ser llamado.

`name`: Nombre del método estático.

`sig`: Descriptor del método. Contiene una nomenclatura específica explicada en el apartado **5.1.5 El tipado de datos en JNI.**

Familia CallStatic<Type>Method: Esta familia de funciones contiene 10 métodos. Estos invocan los métodos estáticos, pertenecientes a otro lenguaje, en este caso Java, utilizando un

identificador. Todos los argumentos del método que se está tratando deben ponerse después del identificador del método.

CallStatic<Type>Method	<Native Type>
CallStaticVoidMethod	void
CallStaticObjectMethod	jobject
CallStaticBooleanMethod	jboolean
CallStaticByteMethod	jbyte
CallStaticCharMethod	jchar
CallStaticShortMethod	jshort
CallStaticIntMethod	jint
CallStaticLongMethod	jlong
CallStaticFloatMethod	jfloat
CallStaticDoubleMethod	jdouble

La signatura general de estos métodos es:

```
<NativeType> CallStatic<Type>Method (jclass clazz, jmethodID methodID, …);
```

`clazz`: Es la referencia a la clase de la cual se invocará el método.

`methodID`: Denota el identificador del método para ser invocado.

Argumentos adicionales: Argumentos que se pasan al método llamado.

Operaciones con cadenas

NewStringUTF: Construye un objeto de tipo Java.lang.String a partir de un array nativo de caracteres de codificación UTF-8.

La signatura de este método es:

```
jstring NewStringUTF(const char *bytes);
```

`bytes`: Puntero a una secuencia de caracteres codificados en UTF-8 que conforman una cadena.

5.1.5 EL TIPADO DE DATOS EN JNI

JNI define un conjunto de tipos C/C++ que se corresponde con los tipos primitivos y de referencia en el lenguaje de programación Java.

Todos los tipos primitivos, tienen una definición correcta de su tamaño en JNI, del mismo modo que ocurre con sus homólogos en Java.

Se muestra en esta tabla la correspondencia entre los tipos primitivos de Java y JNI:

Tipo Java	Tipo nativo (JNI)	Descripción
boolean	**jboolean**	unsigned 8 bits
byte	**jbyte**	signed 8 bits
char	**jchar**	unsigned 16 bits
short	**jshort**	signed 16 bits
int	**jint**	signed 32 bits
long	**jlong**	signed 64 bits
float	**jfloat**	32 bits
double	**jdouble**	64 bits

JNI incluye una serie de tipos de referencia, que se corresponden con diferentes tipos de referencia en el lenguaje de programación Java.

Los tipos de referencia de JNI, están organizados en la jerarquía que se muestra a continuación:

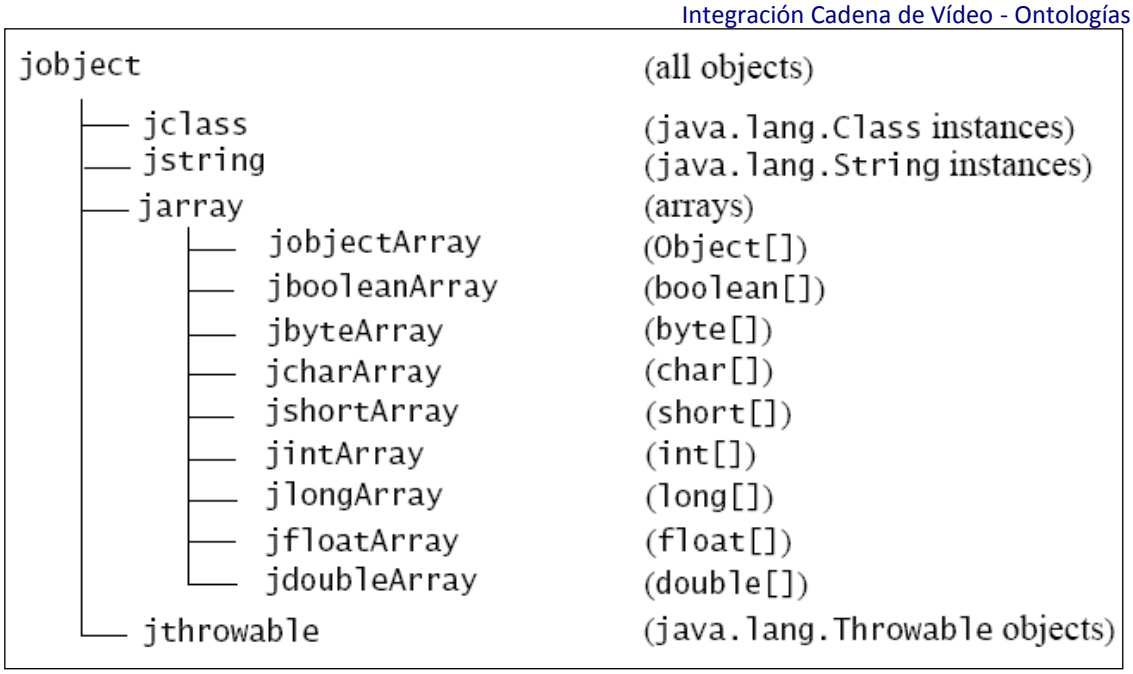

Ilustración 8

Cuando se desea utilizar un tipo de referencia en un lenguaje nativo que no está incluido en esta jerarquía, debe utilizarse jobject para contenerlo.

5.1.6 LOS IDENTIFICADORES DE CLASE, DE CAMPO Y DE MÉTODO EN JNI

Cuando se realizan llamadas desde un código JNI incrustado en un lenguaje nativo (C++) a otro lenguaje (Java), es necesario representar de alguna forma los elementos de ese otro lenguaje usados en esas llamadas, esto es, las clases, los campos y los métodos. Esta forma de representación son los descriptores de JNI.

Descriptores de clase

Un descriptor de clase representa el nombre de una clase o interfaz. Estas pueden venir derivadas de la propia especificación de Java sustituyendo el caracter "." por el caracter "/" en su definición.

Por ejemplo en el caso de la clase Java.lang.String, el descriptor sería "Java/lang/String".

En el caso de las clases estructuradas como arrays, sus descriptores, se conforman usando el caracter "[" seguido del descriptor de campo del tipo de elemento.

Por ejemplo el descriptor utilizado para la clase "int[]" es "[I" y el descriptor utilizado para la clase "double[][][]" es "[[[D"

Descriptores de campo

Estos descriptores son capaces de representar tanto los tipos básicos de datos como los tipos de referencia.

Los descriptores para los 8 tipos primitivos son los siguientes:

Descriptor de campo	Tipo de datos Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double

Los descriptores de los tipos de referencia, empiezan con el caracter “L”, seguido del descriptor de la clase y termina con el caracter ”;”. Los descriptores de campo referidos a arrays siguen la misma norma que en el caso de los descriptores de clases. Se muestran algunos ejemplos de descriptores de campo para tipos de referencia y su correspondencia en el lenguaje Java.

Descriptor de campo	Tipo de datos Java
“Ljava/lang/String;”	String
“[I”	Int[]
“[Ljava/lang/Object;”	Object[]

Descriptores de método

Los descriptores de métodos se conforman colocando los descriptores de campo de todos los tipos de argumentos, entre paréntesis, y después (fuera ya del paréntesis), se define el descriptor de campo del tipo de retorno. No se incluyen espacios u otros caracteres separadores entre los tipos de los argumentos. Para definir un tipo de retorno void, se utiliza “V”. Los constructores deben utilizar “V” como tipo de retorno e “<init>” como nombre.

Aquí se muestran algunos ejemplos de descriptores de métodos JNI y sus correspondientes métodos y métodos constructores.

Descriptor de método	Tipo de datos Java
"()Ljava/lang/String;"	String f();
"(ILjava/lang/Class;)J"	long f(int i, Class c);
"([B)V"	String(byte[] bytes);

Javap

Como esta nomenclatura no es trivial, existe una herramienta Java incluida en cualquier paquete actual de JDK. Esta utilidad se activa con el comando javap y consiste en un desensamblador de ficheros compilados (.class), su salida depende de las opciones utilizadas. En caso de no utilizar opciones, javap imprime los campos y los métodos públicos de la clase pasada por argumento. En todo caso el interés de esta utilidad para este caso, es conocer de forma inmediata la signatura interna de una clase. Esto se consigue incluyendo la opción –s (signature) en la llamada.

Este es un ejemplo de una llamada a javap para obtener las signaturas de la clase Java/lang/Object:

```
javap -s Java/lang/Object
```

Esta sería la traza obtenida a continuación:

```
C:\Users\Mirren de Abia>javap -s java/lang/Object
Compiled from "Object.java"
public class java.lang.Object{
public java.lang.Object();
  Signature: ()V
public final native java.lang.Class getClass();
  Signature: ()Ljava/lang/Class;
public native int hashCode();
  Signature: ()I
public boolean equals(java.lang.Object);
  Signature: (Ljava/lang/Object;)Z
protected native java.lang.Object clone()   throws java.lang.CloneNotSupportedEx
ception;
  Signature: ()Ljava/lang/Object;
public java.lang.String toString();
  Signature: ()Ljava/lang/String;
public final native void notify();
  Signature: ()V
public final native void notifyAll();
  Signature: ()V
public final native void wait(long)   throws java.lang.InterruptedException;
  Signature: (J)V
public final void wait(long, int)   throws java.lang.InterruptedException;
  Signature: (JI)V
public final void wait()   throws java.lang.InterruptedException;
  Signature: ()V
protected void finalize()   throws java.lang.Throwable;
  Signature: ()V
static {};
  Signature: ()V
}
```

Ilustración 9

5.1.7 EJEMPLO DE JNI

En este punto se va a tratar de ilustrar el modo en que se arranca una máquina virtual de Java en una aplicación nativa (Código C/C++). Como ya se indicó anteriormente, la implementación de la JVM (Java Virtual Machine), está alojada en una librería nativa. Las aplicaciones nativas pueden enlazar con esta biblioteca y usar la interfaz de invocación para cargar la máquina virtual.

La forma más inmediata de entender esto es mostrar un ejemplo de este caso para una llamada a un método estático de una clase Java que no requiere la creación de ningún objeto.

La susodicha clase se llama Cuenta, esta a su vez, tiene un método estático, llamado cuenta que devuelve el número de archivos que contiene un fichero con extensión .zip o un cero en

```
import java.util.zip.*;
import java.io.IOException;
public class Cuenta {
   static int cuenta() {
        try {
                ZipFile f = new ZipFile("miFichero.zip");
                return f.size();
        } catch (IOException ex) {
                return 0;
        }
   }
}
```

Ilustración 10

caso de excepción. Este sería el código de esa clase.

Para poder ejecutar esta clase, es necesaria la creación de una máquina virtual a partir del método init.

Para poder llamar al método deseado, es necesario obtener un manejador de la clase que lo contiene. Para ello se utiliza el método FindClass. El argumento tomado por este método, es el nombre de la clase que contiene el código que se desea ejecutar.

Una vez cargada la clase en la JVM, hay que obtener una referencia del método a llamar, para ello, se utiliza GetStaticMethodID. La función recibe el identificador de la clase que contiene el método, el nombre del método (en nuestro ejemplo, cuenta), y el tipo del método. Para

indicar el tipo se utiliza una cadena con una codificación especial, que permite representar cualquier tipo de Java [3]. Para conocer las codificaciones de los métodos de forma sencilla, existe un ejecutable en todos los paquetes Java, denominado javap, a partir del cual se pueden obtener.

Para llamar definitivamente al método estático, se utiliza la función CallStaticIntMethod que recibe los manejadores de la clase y el método y devuelve el valor que retornaría el método llamado.

```
#include "JavaIntfz.h"
int main(int argc, char* argv[]) {
    // 1.- Creación de la máquina virtual de Java
        JavaIntfz::JVM jvm;
        jvm.init(".");
    // 2.- Conseguir la referencia a la clase Cuenta
        jclass cls;
        cls = jvm->FindClass("Cuenta");
        if (cls == 0) {
                fprintf(stderr, "Clase 'Cuenta' no encontrada.\n");
                return -1;
        }
    // 3.- Conseguir la referencia al método int cuenta().
        jmethodID mid;
        mid = jvm->GetStaticMethodID(cls, "cuenta", "()I");
        if (mid == 0) {
                fprintf(stderr, "Can't find Cuenta.cuenta\n");
                return -1;
        }
    // 4.- Llamada al método, y escritura de resultados.
        printf("%d", jvm->CallStaticIntMethod(cls, mid));
        return 0;
}
```

5.2 LA INTERFAZ DE COMUNICACION

Las capas en las que está basada esta aplicación de tracking, están unidas, a través de una interfaz de comunicación que garantiza la interoperabilidad y la independencia entre ellas.

Esta interfaz tiene un doble objetivo:

- Es utilizada para actualizar el modelo de la capa de contexto de acuerdo a las informaciones capturadas por la capa de seguimiento ya sea en las fases de actualización, creación o eliminación de los tracks. La información de cada track, pasa a la interfaz en forma de datos en formato numérico y es la interfaz la que tiene el deber de transformar estos datos en instancias ontológicas
- Es el mecanismo a través del que se realizan las consultas acerca de las recomendaciones razonadas por la capa de contexto.

Puede observarse, que la capa de seguimiento es la que a partir de las llamadas a la interfaz activa los procesos de la capa de contexto, pero en el caso de las recomendaciones, es la capa de contexto, la que debe iniciar la comunicación para devolver las respuestas. Esto permite que se mantenga cierto sincronismo entre ambas capas, si la capa de seguimiento no esperase las recomendaciones de la capa de contexto en cada frame, esta última realizaría las recomendaciones mucho o por lo menos, algún tiempo después (lo que se traduce en frames después), del momento en que eran verdaderamente necesarias.

En todo caso, existe un modo de ejecución asíncrono, en el cual, las llamadas a la consulta de recomendaciones se pueden hacer en cualquier momento. Para implementar esta asincronía, existe en la interfaz, una cola de recomendaciones, que va siendo rellenada por la capa de contexto a medida que el razonador deduce las mejoras que pueden llevarse a cabo. La capa de seguimiento recupera las recomendaciones de la cola de acuerdo a una política de prioridades.

5.2.1 LOS MÉTODOS DE LA INTERFAZ DE COMUNICACIÓN

Esta interfaz de comunicación consta de una serie de métodos heredados de una interfaz de programación denominada ContextModel. Estos son los que permiten la comunicación entre las capas de las que consta la aplicación.

Se muestran de forma resumida sus signaturas y cometidos:

```
void frameCreated(int newFrameID);
```

Añade un nuevo frame al ContextModel indicando su identificador. Esta operación se repite cada vez que se analiza un nuevo frame.

```
void trackCreated(int newTrackID, int frameID);
```

Añade un nuevo track al ContextModel indicando su identificador y el frame en el que se reconoció.

```
void trackDeleted(int trackID, int frameID);
```

Declara un track como inactivo desde que es eliminado de la escena por la capa de seguimiento.

```
void trackUpdated(int trackID, int frameID, float w, float h, float posX, float posY, float vXPred, float vYPred);
```

Actualiza las propiedades de cada track que hayan cambiado en cada frame. Las propiedades utilizadas actualmente son el ancho y el alto del track, su posición determinada por una coordenada (x,y) y la predicción acerca de su velocidad en cada uno de los planos x e y.

A pesar de que no se ha hablado de ello anteriormente existen métodos en esta interfaz, no utilizados en el desarrollo actual, que otorgan una mayor capacidad de expresividad de la capa de tracking hacia la capa de contexto, estos son:

```
void groupTrack(int mainTrackID, int groupedTrackID, int frameID);
```

Indica que un track (groupedTrackID) se agrupa con otro track (mainTrackID) en la escena. Este método crea una entrada asociada al track (groupedTrackID).

```
void occludedTrack(int mainTrackID, int occludedTrackID, int frameID);
```

Indica que un track (occludedTrackID) es ocultado por otro track (mainTrackID) en la escena. Este método crea una entrada asociada al track (occludedTrackID).

```
void unGroupTrack(int mainTrackID, int grupedTrackID, int frameID);
```

Indica que un track (groupedTrackID) se desagrupa de otro track (mainTrackID) en la escena. Este método crea una entrada asociada al track (groupedTrackID).

```
unOccludeTrack(int mainTrackID, int occludedTrackID, int frameID);
```

Indica que un track (occludedTrackID) deja de ser ocultado por otro track (mainTrackID) en la escena. Este método crea una entrada asociada al track (occludedTrackID).

5.3 EL WRAPPER

Para llevar a cabo la comunicación entre capas, es necesario realizar una traducción de la información de los tracks, de C++, que se corresponde con el lenguaje de la capa de seguimiento; a Java lenguaje en el que se basa la capa de contexto.

Para ello, se ha desarrollado un wrapper de modo que a partir del código nativo de la aplicación, es posible utilizar la interfaz de comunicación, explicada en el apartado anterior.

Esto quiere decir que el código que realiza las llamadas a la interfaz de comunicación, está incrustado dentro de la capa de seguimiento y utiliza la tecnología JNI a partir de código C++.

5.3.1 ¿CÓMO ES POSIBLE UTILIZAR JAVA A TRAVÉS DE C++?

Desde la versión 1.1 de la herramienta JDK, la implementación de la máquina virtual de Java está incluida en el paquete como una librería dinámica, es decir, una DLL en caso de Windows o un archivo con extensión .so en caso de Linux. Gracias a esto, es posible añadir a una

aplicación C/C++ (aplicación nativa), la máquina virtual de Java, llamando a funciones contenidas en esa DLL [3]. De hecho, la forma normal de ejecutar la máquina virtual, (el ejecutable Java), es un programa de C que interpreta los argumentos de la línea de comandos, y llama a la máquina virtual utilizando esa DLL que implementa el interfaz de JNI.

En un ámbito más técnico y para orientar futuros desarrollos hay que destacar, que es necesario incluir en el enlazado del proyecto C++ que se está desarrollando, el archivo jvm.lib (que se encuentra en el directorio $JDKROOT$\lib), esto provoca que el sistema operativo cargue la DLL en el momento de la carga del ejecutable, también es necesario que el preprocesador de C++ sea capaz de encontrar las cabeceras funciones JNI a partir del archivo jni.h (que se encuentra en el directorio $JDKROOT$\include) y jni_md.h (que se encuentra en el directorio $JDKROOT$\include\win32).

Por otro lado, como la DLL que contiene la implementación de la máquina virtual, debe ser encontrada en el momento de la ejecución de la aplicación, es necesario incluir en el PATH del sistema el lugar donde se encuentra dentro del Java Development Kit. En los paquetes JDK actuales existen 2 implementaciones de la máquina virtual, una se utiliza para ejecutar aplicaciones en un sistema cliente y otra para ejecutarlas en sistemas servidores. Este proyecto en particular a utilizado la implementación cliente (que se encuentra en el directorio $JDKROOT$\jre\bin\client).

5.3.2 ¿CÓMO SE HACE?

Desde un punto de vista general, la implementación de un wrapper, según las condiciones de este proyecto, debe comenzar creando una máquina virtual a partir de una llamada a una función de JNI. Después es el momento de cargar las clases que se vayan a utilizar durante la ejecución, y realizar las llamadas a los métodos específicos de las clases cargadas, como si de una aplicación normal se tratase (si bien es cierto que las llamadas no tienen una sintaxis trivial). Por último, una vez ejecutado el código Java, debería destruirse la máquina virtual liberando así la memoria del computador.

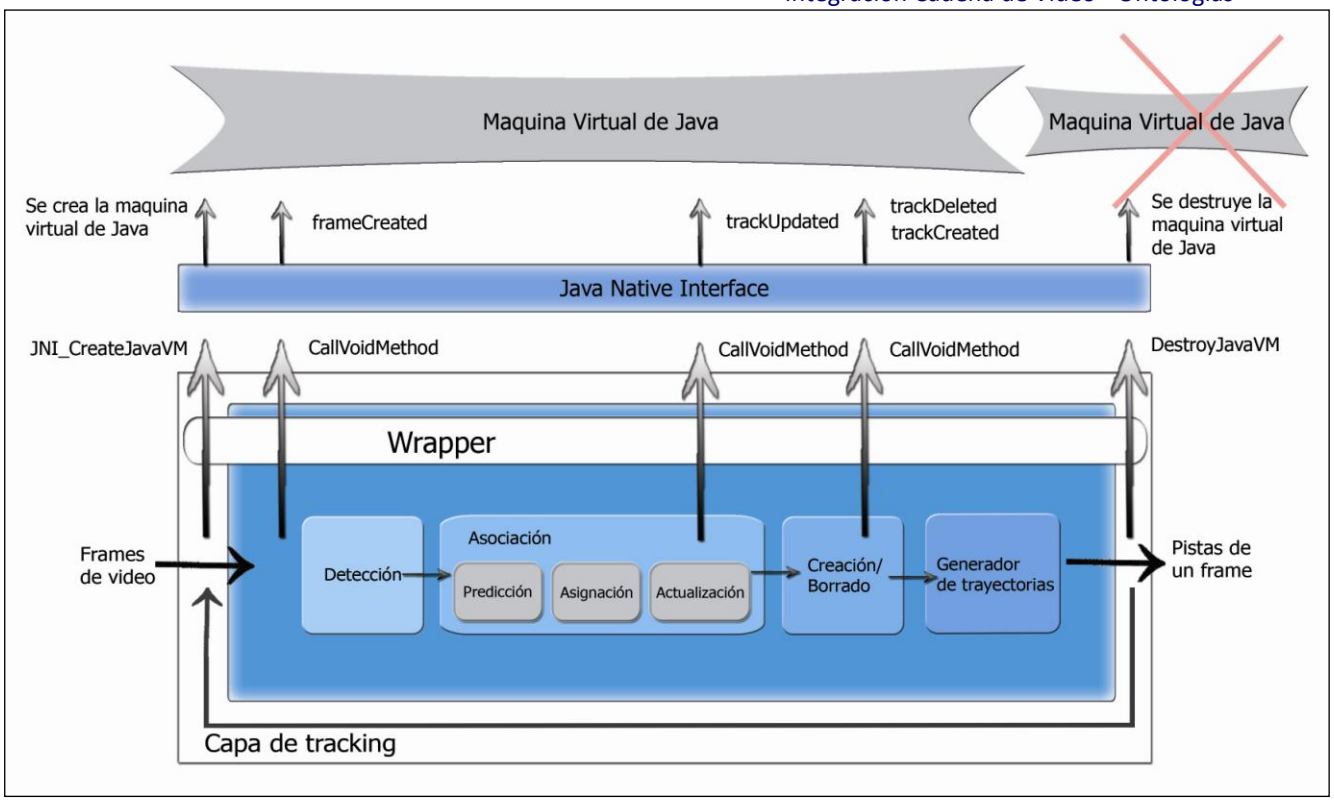

Ilustración 11

Debido a que el proceso de creación de una JVM es costoso en tiempo y recursos, es preferible arrancarla al inicio del programa y destruirla una vez se han finalizado todas las operaciones realizadas en Java.

5.3.3 ¿CÓMO FUNCIONA EL WRAPPER?

A pesar de lo dicho en el punto anterior, en la práctica, el wrapper realiza una comprobación inicial de la versión de JNI utilizada, debido a que la inicialización de la máquina virtual debe realizarse de forma específica, de acuerdo con su versión.

Como se vio en el apartado 5.1.3 Funciones de JNI, la máquina virtual se inicializa a través del comando, `JNI_CreateJavaVM`.

Para invocar esta función además de los punteros JavaVM y JNIEnv, que antes de la llamada están vacios, es necesario pasar por argumento, el directorio de las librerías cuyas clases van a ser utilizadas durante la ejecución, dependiendo de la versión, se utilizaran objetos de tipo `JavaVMInitArgs` (versión 1.2 y posteriores) o `JDK1_1InitArgs` (versión 1.1), para contener estos directorios.

Una vez se ha creado la máquina virtual y antes de comenzar a realizar las llamadas a la interfaz de comunicación, es necesario crear los objetos que la soportan. La interfaz en el que

se basa el desarrollo para la creación de estos objetos es, JNIEnv, inicializada tras levantarse la máquina virtual, en el código explicado posteriormente esta interfaz está representada por un puntero llamado `env`. Desde un punto de vista amplio, el wrapper puede dividirse en 3 secciones:

1. Creación del objeto ContextModelManager: Crea un objeto de tipo ContextModelManager, a través de una llamada a un método estático (en el punto **5.1.4 La interfaz JNIEnv**, se indica que JNI utiliza métodos y estrategias diferentes para abordar las diferentes llamadas de C++ a Java).
2. Obtención del objeto ContextModel: Como el objeto ContextModelManager contiene el modelo de contexto, únicamente, es necesario llamar a un método "get" que entrega un objeto de tipo ContextModel.
3. Las llamadas a la interfaz de comunicación: El objeto obtenido de tipo ContextModel se encarga de soportar la interfaz de comunicación, es decir, las llamadas de creación de cada frame y la creación, borrado y actualización de todas las pistas de la escena en las ontologías.

A continuación, se muestra un diagrama pormenorizado de la ejecución del wrapper en la aplicación, debido a su complejidad, en cada sección se ha incluido un diagrama propio que ilustra los pasos dados en cada caso.

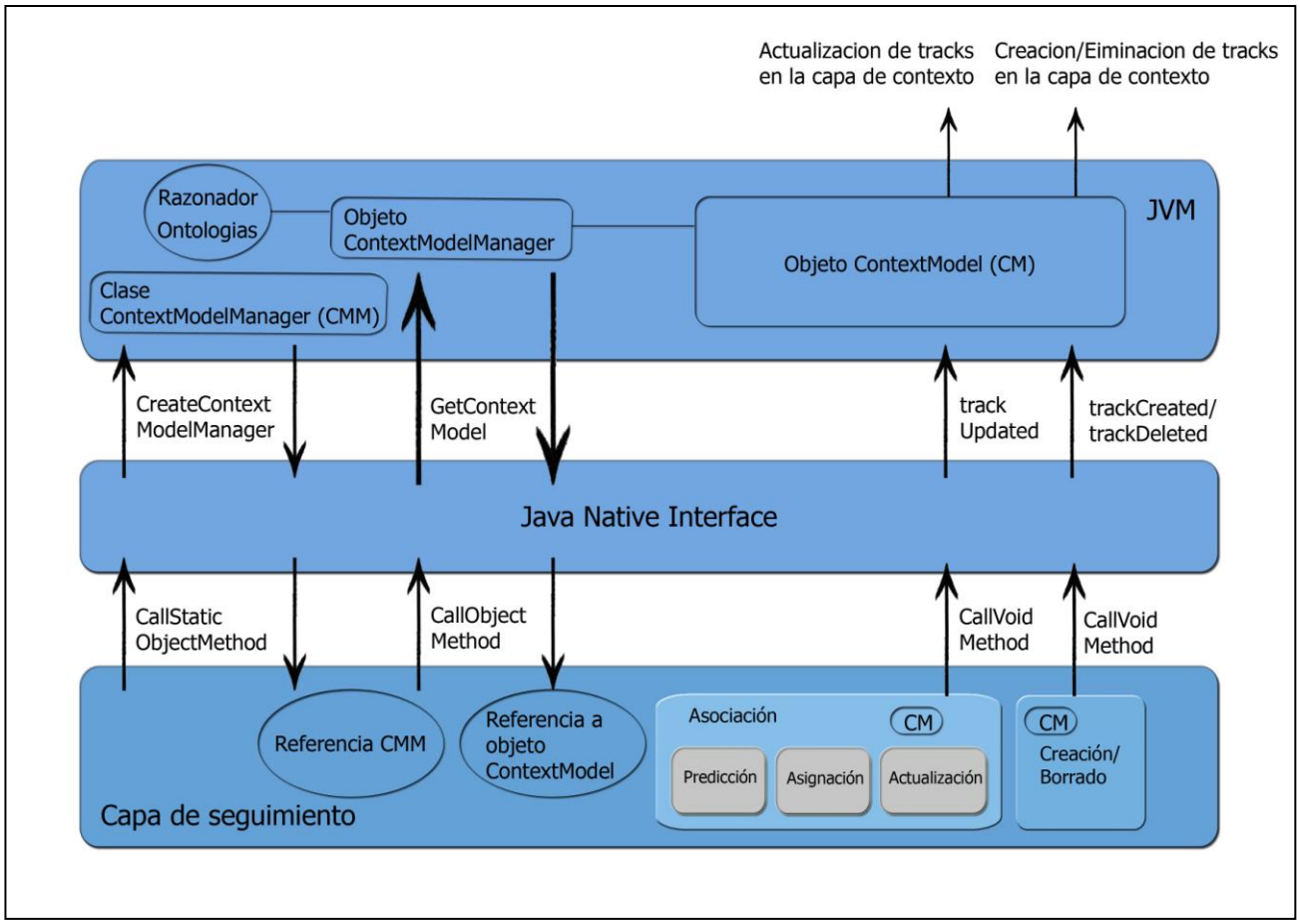

Ilustración 12

Desde una perspectiva más concreta, para que el wrapper realice estas 3 fases, es necesario que realice múltiples operaciones.

1. Creación del objeto ContextModelManager:

ContextModelManager, es una pseudofactoría de elementos que se utilizan en la ejecución, entre otras cosas, esta clase contiene el razonador y las ontologías. Es necesario tener una instancia de este tipo para, a partir de ella, obtener un objeto que represente el modelo de contexto.

Para obtener una instancia de esta clase, es necesario realizar una llamada al método estático `createContextModelManager` de la misma, que devuelve un objeto de tipo ContextModelManager.

Para poder realizar esta llamada, el primer paso es obtener una referencia a la clase que lo contiene, el tipo de esta referencia es jclass, definido en jni.h. Para conseguirlo, se utiliza el método `FindClass`, este método recibe el nombre de la clase que se desea y la carga en la máquina virtual de Java.

```
jclass cls = env->FindClass("contextmodel/ContextModelManager");
```

Cuando se realizan estas llamadas, deben indicarse, los paquetes a los que contiene cada clase, como puede observarse, la clase ContextModelManager está incluida en el paquete contextmodel.

Posteriormente, debe realizarse algo similar pero, en este caso, con el método a llamar `createContextModelManager`. El manejador de este, se obtiene utilizando el método de JNI, `GetStaticMethodID`.

```
jmethodID mid = env->GetStaticMethodID(cls,
"createContextModelManager",
"(Ljava/net/URI;)Lcontextmodel/ContextModelManager;");
```

La función, recibe como argumentos, el identificador de la clase que contiene el método (`cls`), el nombre del método y el tipo, cuya codificación puede consultarse en el punto **5.1.6 Los identificadores de clase, de campo y de método en JNI**.

Como la llamada a este método, contiene un objeto de tipo URI, es necesario inicializarlo. Según el API de Java, la clase URI proporciona un medio simple y extensible para la identificación de un recurso. El cometido de este objeto es apuntar a un documento que contiene los datos referentes a las ontologías utilizadas en la ejecución, el documento, contiene tanto una dirección local, que es utilizada como primera opción, así como una dirección de Internet que se utiliza como recurso alternativo. Como en el caso anterior, el procedimiento, también es obtener el manejador de la clase a través del método FIndClass.

```
jclass URIClass = env->FindClass("Java/net/URI");
```

Después se debe obtener el manejador del método que instancia la clase convirtiendola en un objeto, es decir, el manejador del contructor, utilizando `<init>`.

```
jmethodID URIMid= env->GetMethodID(URIClass, "<init>",
"(Ljava/lang/String;)V");
```

Debido a que el constructor de esta clase utiliza una cadena de caracteres como argumento, que es el archivo de configuración que contiene la ubicación de las ontologías, debe construirse un string y contenerlo en un tipo nativo (jstring). Esto puede hacerse a través del método `NewStringUTF` que codifica la cadena de caracteres en formato UTF-8 para que sea válida en Java.

```
jstring configpath = env->
NewStringUTF("contextmodeltest/config/config.xml");
```

Una vez se tienen todos los ingredientes, debe instanciarse el objeto mediante la llamada `NewObject`.

```
jobject arg = env->NewObject(URIClass, URIMid, configpath);
```

Los atributos pasados a esta llamada, para crear este nuevo objeto, son: el descriptor de la clase que se desea instanciar, el descriptor del método constructor y el argumento, tipado como un dato Java, del constructor de la clase URI. El resultado final es un objeto de tipo URI que apunta al documento indicado en `configpath`.

La obtención definitiva de un objeto de tipo ContextModelManager, se realiza a través de la llamada `CallStaticObjectMethod`. Esta llamada se utiliza específicamente para los métodos estáticos de una clase como ya se explico en el punto **5.1.4 La interfaz JNIEnv.**

```
jobject ctxModelMan = env->CallStaticObjectMethod(cls, mid, arg);
```

Los argumentos de este método son: el manejador de la clase instanciada en la variable `cls`, el método estático llamado `createContextModelManager`, cuyo manejador esta instanciado en `mid`, también y el argumento del método llamado `arg`, es decir, el objeto de tipo URI.

Para simplificar al lector esta relación entre el código nativo y el código Java se muestra en forma de diagrama esta primera fase, se puede comprobar que se corresponde con la fase inicial del diagrama general presentado anteriormente.

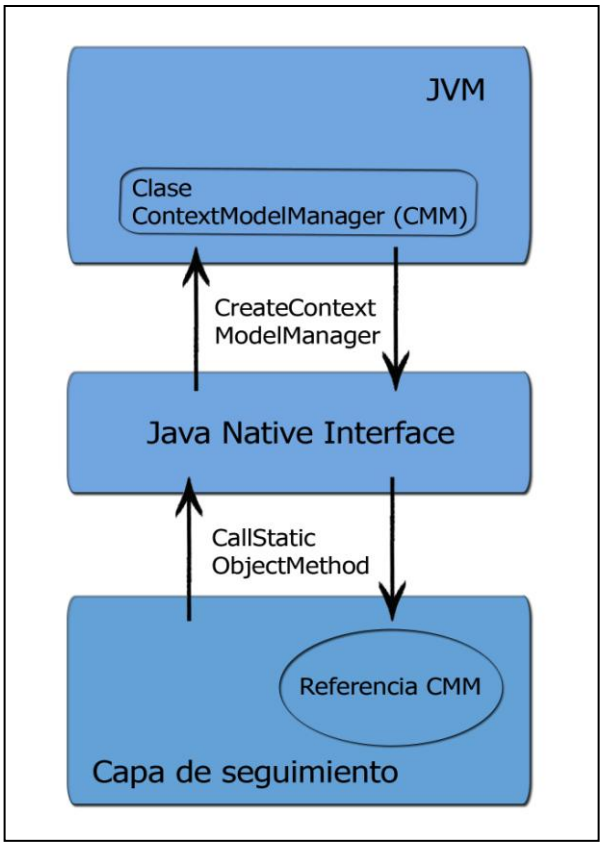

Ilustración 13

2. Obtención del objeto ContextModel:

Una vez se ha conseguido el gestor de las ontologías, se debe obtener a partir de él, un objeto del tipo ContextModel que actuara como interfaz de comunicación. Esta clase, es la que contiene los métodos explicados en el punto **5.2.1 Los métodos de la interfaz de comunicación**.

Se comienza este paso, por tanto, consiguiendo el manejador del método `getContextModel`, el valor devuelto por este método, debe ser un objeto de tipo ContextModel.

```
jmethodID ctxModelManMid = env->GetMethodID(cls, "getContextModel",
"()Lcontextmodel/ContextModel;");
```

En esta ocasión no es necesario incluir en el procedimiento, la obtención del manejador de la clase, ya que el método llamado, se encuentra en ContextModelManager (descriptor `cls`, heredado de operaciones anteriores). Los otros 2 argumentos son, el nombre del método usado y su tipo o signatura, cuya definición puede consultarse en el punto **5.1.6 Los identificadores de clase, de campo y de método en JNI.**

Una vez se tiene el descriptor del método instanciado en la variable `ctxModelManMid`, se realiza la llamada `CallObjectMethod` que llama a un método contenido en un objeto.

```
jobject ctxModel = env->CallObjectMethod(ctxModelMan, ctxModelManMid);
```

El objeto pasado por argumento en este caso, es el gestor de ontologías, instanciado en la variable `ctxModelMan`.

El objeto de tipo ContextModel obtenido finalmente y que como se ha indicado anteriormente, soporta la interfaz de comunicación, es en realidad un objeto del subtipo SimpleContextModel que implementa la interfaz ContextModel. Esto no cambia nada la perspectiva existente, pero sirve para aclarar que ContextModel, es una interfaz que puede ser implementada por diferentes modelos de contexto.

En el siguiente diagrama se muestra de forma simplificada esta implementación, es posible comprobar cómo esta segunda fase se corresponde con la segunda fase del diagrama general expuesto anteriormente.

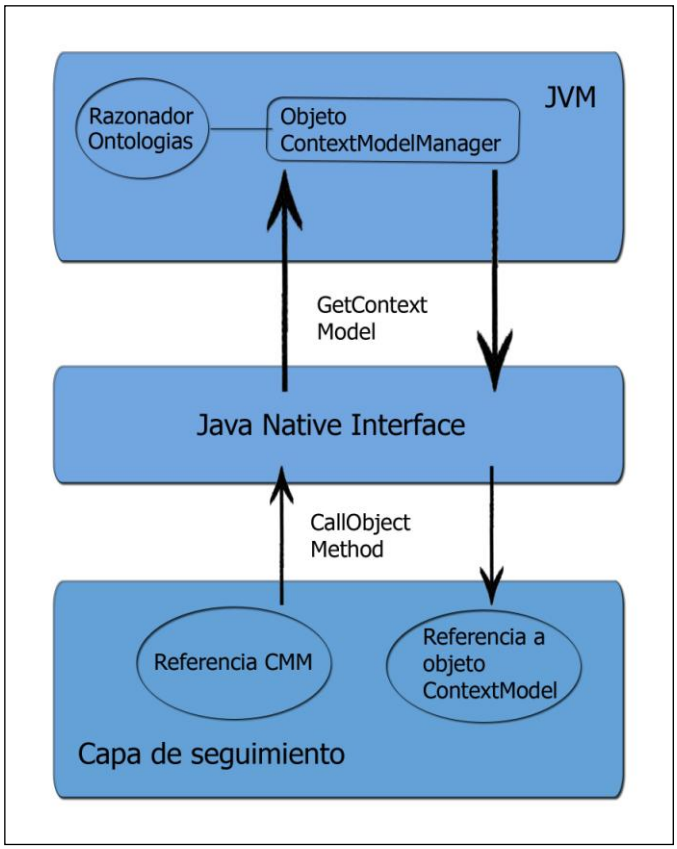

Ilustración 14

3. Las llamadas a la interfaz de comunicación:

Las diferentes llamadas a la interfaz de comunicación, se realizan de una forma ordenada según el módulo en el que se encuentra la ejecución del tracker.

Es necesario crear un frame en el modelo de representación, en cada ciclo del tracker, después de cada análisis de imagen, antes de realizar cualquier operación básica (crear, eliminar, o actualizar pistas). Esta representación simbólica de un frame se utiliza para mantener la coherencia temporal, de este modo se puede realizar un análisis desde la perspectiva de un frame, en vez de conocer la vida de un track por sus frames se puede conocer que cosas ocurrieron en cada frame.

Para crear un frame en la capa de contexto, a través de la interfaz de comunicación, se hace uso del mecanismo clásico en JNI. Primero, se llama al método `GetMethodID`, para obtener el descriptor del método `frameCreated`. Este método se encuentra en la clase ContextModel, cuyo descriptor está guardado en `ctxModelClass`.

```
jmethodID ctxModelMid = env->GetMethodID(ctxModelClass,
"frameCreated", "(I)V");
```

A partir del descriptor del método y a partir del objeto ya instanciado de la clase (`ctxModel`), se hace la llamada deseada utilizando la función JNI, `CallVoidMethod`.

```
env->CallVoidMethod(ctxModel, ctxModelMid, (jint)m_FrameCount);
```

Este método nativo, incluye como argumento final, el número de frame (`m_FrameCount`). Como puede verse en el código ha sido necesario realizar una conversión por casting explícito, desde el tipo entero en C++ al tipo correspondiente de entero en Java. El punto **5.1.5 El tipado de datos en JNI** amplía la información acerca del tipado de datos en JNI.

Existe una fase de actualización, cuando la ejecución llega al módulo de asociación, una vez completada la creación de un frame en cada ciclo. En esta fase, se ha añadido el código de llamada a la interfaz de comunicación para actualizar los tracks en las ontologías. Esta llamada actualiza todos los atributos referentes a las pistas que tengan nueva actividad, respecto a la imagen anterior. Esta actividad se refiere a la modificación de sus parámetros durante el análisis de vídeo. La actualización, debe extenderse, a todos los niveles de la estructura ontológica y provocar cambios en el modelo representado, y en consecuencia en la salida de la capa de contexto hacia la capa de seguimiento, es decir, en las recomendaciones finales.

Para realizar esta llamada, como de costumbre en JNI, se obtiene el descriptor del método `trackUpdated`, a partir de la clase que contiene el modelo de contexto.

```
jmethodID ctxModelUpd = env->GetMethodID(ctxModelClass,
"trackUpdated", "(IIFFFFFF)V");
```

Con el descriptor obtenido, se realiza la llamada al método `CallVoidMethod`, sobre el objeto que contiene la interfaz de comunicación (`ctxModel`).

```
env->CallVoidMethod(ctxModel, ctxModelUpd, (jint)CV_BLOB_ID(pB),
(jint)m_FrameCount, (jfloat)pB->w, (jfloat)pB->h, (jfloat)pB->x,
(jfloat)pB->y, (jfloat)0, (jfloat)0);
```

Como puede observarse, tras el descriptor del objeto de tipo ContextModel (`ctxModel`) y el descriptor del método (`ctxModelUpd`), se pasan todos los parámetros usados por el método Java; el identificador de la pista (`CV_BLOB_ID(pB)`), el frame en el que sufre la modificación la pista (`m_FrameCount`), la anchura (`pB->w`) y altura de la pista (`pB->h`) y su posición asociada (`pB->x, pB->y`) en un plano de 2 dimensiones.

El resto de parámetros, están reservados para usos futuros y tienen que ver con la velocidad de la pista dividida en sus componentes 'vx' y 'vy' del plano.

Una vez más ha sido necesario, realizar una conversión por casting explícito de los argumentos del método, desde un tipo primitivo en C++ (float e int) a un tipo correspondiente en Java (jfloat y jint).

En el módulo de inicialización y borrado del tracker, se crean y eliminan las pistas del tracker, es por tanto el lugar escogido para realizar las llamadas apropiadas a la interfaz de comunicación y crear o eliminar las pistas de forma simbólica en la capa de contexto.

Puede observarse en el código siguiente, cómo, de manera casi análoga, se crean los descriptores de los métodos, de la interfaz de comunicación, de borrado (`trackDeleted`) y creación (`trackCreated`) de pistas. En ambos casos se hace uso del descriptor de la clase sin instanciar de ContextModel, contenida en `ctxModelClass` y de la signatura del método.

```
jmethodID ctxModelDel = env->GetMethodID(ctxModelClass,
"trackDeleted", "(II)V");

jmethodID ctxModelNew = env->GetMethodID(ctxModelClass,
"trackCreated", "(II)V");
```

La llamada a la interfaz de comunicacion, se hace en los dos casos, utilizando los mismos argumentos e invocando el método en la interfaz de comunicación (`ctxModel`).

La unica diferencia entre las llamadas, es el uso de descriptores correspondientes a métodos diferentes, uno para crear pistas simbólicas en las ontologias de la capa de contexto y otro para borrarlas.

```
env->CallVoidMethod(ctxModel, ctxModelDel, (jint)CV_BLOB_ID(pB),
(jint)m_FrameCount);
env->CallVoidMethod(ctxModel, ctxModelNew, (jint)CV_BLOB_ID(pB),
(jint)m_FrameCount);
```

Los argumentos pasados al método Java (que utilizan una vez más un casting explícito), se corresponden con el identificador de la pista (`CV_BLOB_ID(pB)`) y el frame (`m_FrameCount`) en el que la pista sufre el proceso de creación o de eliminación.

En el siguiente diagrama se muestra la implementación de la fase final del wrapper, esta última fase se corresponde con la parte terminal del diagrama general expuesto anteriormente.

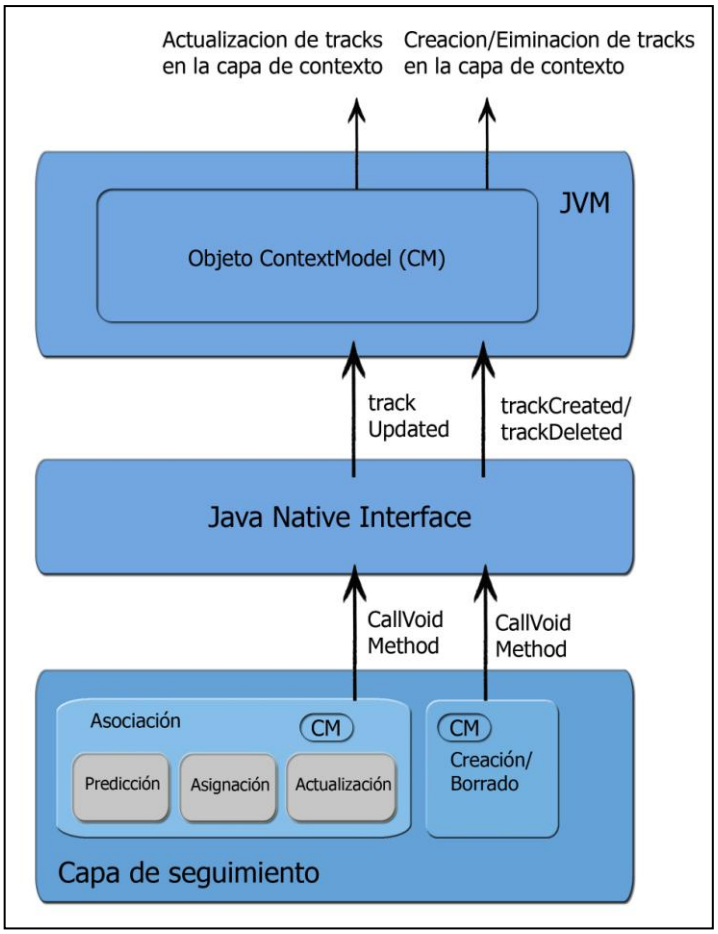

Ilustración 15

Una vez se ha realizado la completa ejecución de todos los ciclos (1 ciclo por frame) de análisis de vídeo debe destruirse la máquina virtual para no desaprovechar los recursos de la computadora, esto se hace mediante la siguiente línea.

```
jvm->DestroyJavaVM();
```

A pesar de que no se han documentado explícitamente, existen muchos posibles errores en la ejecución de la especificación JNI. Por ejemplo, la incapacidad de la herramienta para encontrar una clase determinada y cargarla en la máquina virtual a través del método

FindClass. Este tipo de situaciones, provocan el salto de la aplicación, mediante un comando `goto <etiqueta>`, al código que contiene la llamada a la destrucción de la JVM.

6 PRUEBAS DE CONCEPTO

Este capítulo trata de explicar al lector los resultados obtenidos a partir de la experimentación con el sistema. En primera instancia se realiza una introducción de la prueba analizada y de las herramientas utilizadas para llevarla a cabo, en segundo lugar se introduce la estructura de la ontología empleada para las prueba, en tercer término se examinan los resultados obtenidos a partir del análisis de un vídeo y para concluir se observa detalladamente el coste temporal de la inclusión de la capa de contexto.

6.1 INTRODUCCIÓN

El objetivo del análisis de los datos, no es otro que demostrar que la aplicación obtiene una representación ontológica, a partir del la realización de un procesado de seguimiento de un vídeo.

No es necesario usar más de un vídeo analizado para comprobar que el sistema de seguimiento pasa su información al sistema de contexto y que este construye una representación simbólica de los datos obtenidos a través del seguimiento.

El vídeo utilizado ha sido escogido al azar de entre videos de corta duración con extensión .avi. El propósito de la corta duración del vídeo es evitar una gran complejidad en el análisis de la ontología obtenida y mejorar los tiempos de las pruebas realizadas que en algunos casos pueden llegar a ser realmente largos. Para disminuir la complejidad, también se ha tratado de tomar un vídeo con poca actividad, es decir, pocos tracks para que el volumen de información actualizada en cada ciclo del análisis no fuese excesivo y por tanto el tiempo por prueba no pudiera dispararse.

Las herramientas utilizadas para llevar a cabo el análisis de los resultados obtenidos son de carácter gráfico, en ambos casos se tratan de plugins de la aplicación Protégé:

- OWL Viz: El objetivo del uso de este plugin es mostrar de forma gráfica el diagrama de conceptos de la ontología diseñada de forma previa a la ejecución. Gracias a su funcionalidad es posible dividir los diagramas y visualizarlos según los conceptos que interesen en cada ocasión.
- Jambalaya: Este plugin se ha utilizada para verificar el análisis de los resultados. Esta aplicación ha llevado a cabo la visualización de los datos inferidos, esto incluye los individuos creados pertenecientes a los conceptos y las relaciones existentes entre ellos durante la ejecución del análisis.

6.2 ESTRUCTURA DE LA ONTOLOGÍA A ANALIZAR

El modelo actual de la aplicación de tracking de vídeo, contiene 3 ontologías fundamentales:

- Geometry: Es una ontología de conceptos geométricos, obtenida de un desarrollo previo y modificada para ser usada en esta aplicación.
- OWL-Time: Es una ontología de conceptos temporales desarrollada por el World Wide Web Consortium (W3C).
- Trend: Esta ontología ha sido específicamente diseñada para representar los conceptos relacionados con visión por computador provenientes directamente del seguimiento.

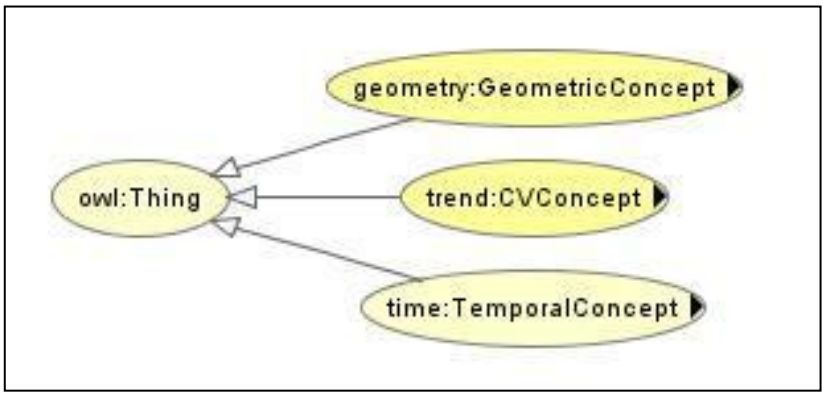

Ilustración 16

Como puede observarse en la ilustración todas estas ontologías parten de la ontología OWL cuyo concepto principal es Thing y a partir de esta desarrollan su modelo específico.

6.2.1 ONTOLOGÍA GEOMETRY

Geometry fue obtenida a partir de un intento de estandarizar las ontologías para la visión por computador y consistía en una de las 3 partes en las que se dividía esta tentativa. Posteriormente fue modificada para ser usada en este caso particular.

Geometry trata de describir el dominio de los objetos desde una perspectiva espacio-temporal, la jerarquía, está formada por conceptos geométricos que pueden ser usados para describir la forma de los objetos del dominio. Como es habitual las ontologías se han construido desde los conceptos más básicos a los conceptos más complejos formados a partir de un conjunto de los básicos.

El tamaño de un objeto puede ser descrito por un cuantificador y este a su vez puede ser descrito por un conjunto de cuantificadores. Es necesario tener en cuenta que la cuantificación se puede hacer de una manera absoluta o relativa a otro concepto [10].

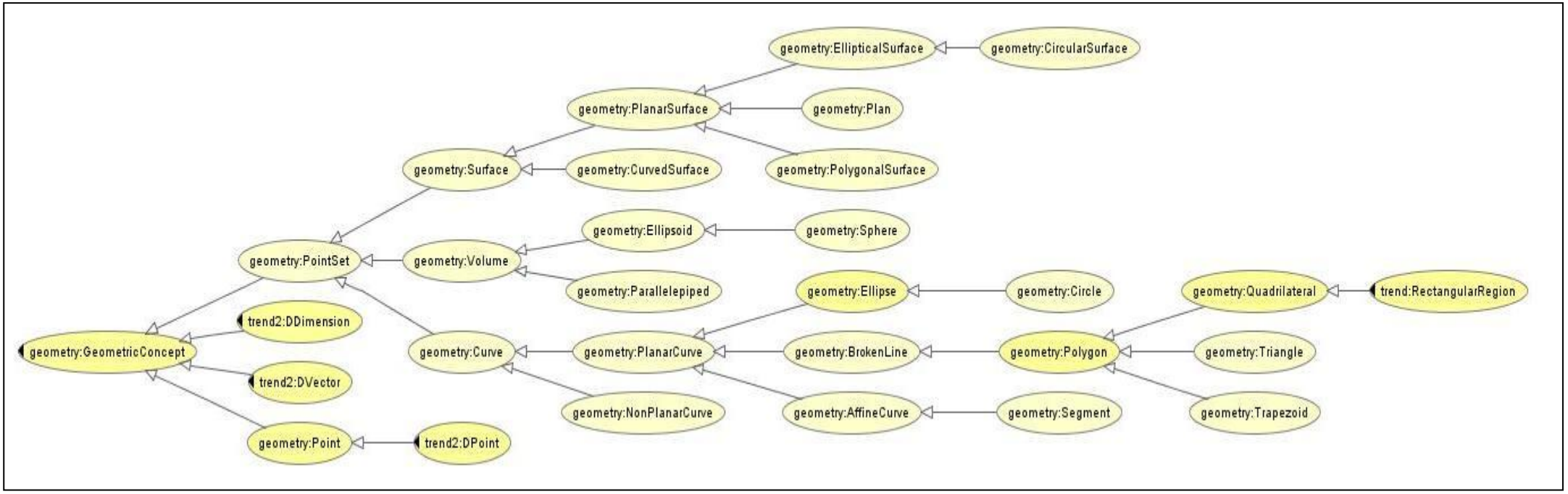

Ilustración 17

6.2.2 ONTOLOGÍAS OWL-TIME Y TIME

OWL-Time fue creada para describir el contenido temporal de las páginas web y las propiedades temporales de los servicios web.

Esta ontología proporciona un lenguaje para expresar hechos a través de las relaciones topológicas entre instantes e intervalos, junto con información acerca de la duración y la fecha y hora.

Como puede observarse en la imagen de la página siguiente, donde se observan todos los conceptos relativos a esta ontología; Time describe entidades temporales, a partir de las cuales es capaz de conformar instantes (Instant) e intervalos (Interval). De forma casi intuitiva se puede entender que un instante, se refiere a un momento del tiempo único, sin puntos de tiempo intermedios y un intervalo es un conjunto de instantes. Según esta definición, un instante sería un intervalo de longitud cero con el mismo principio y fin.

Existen además clases (DurationDescription y DateTimeDescription) con diferentes conjuntos de relaciones para las descripciones temporales.
Para una especificación detallada de esta ontología puede visitarse
http://www.w3.org/TR/owl-time/

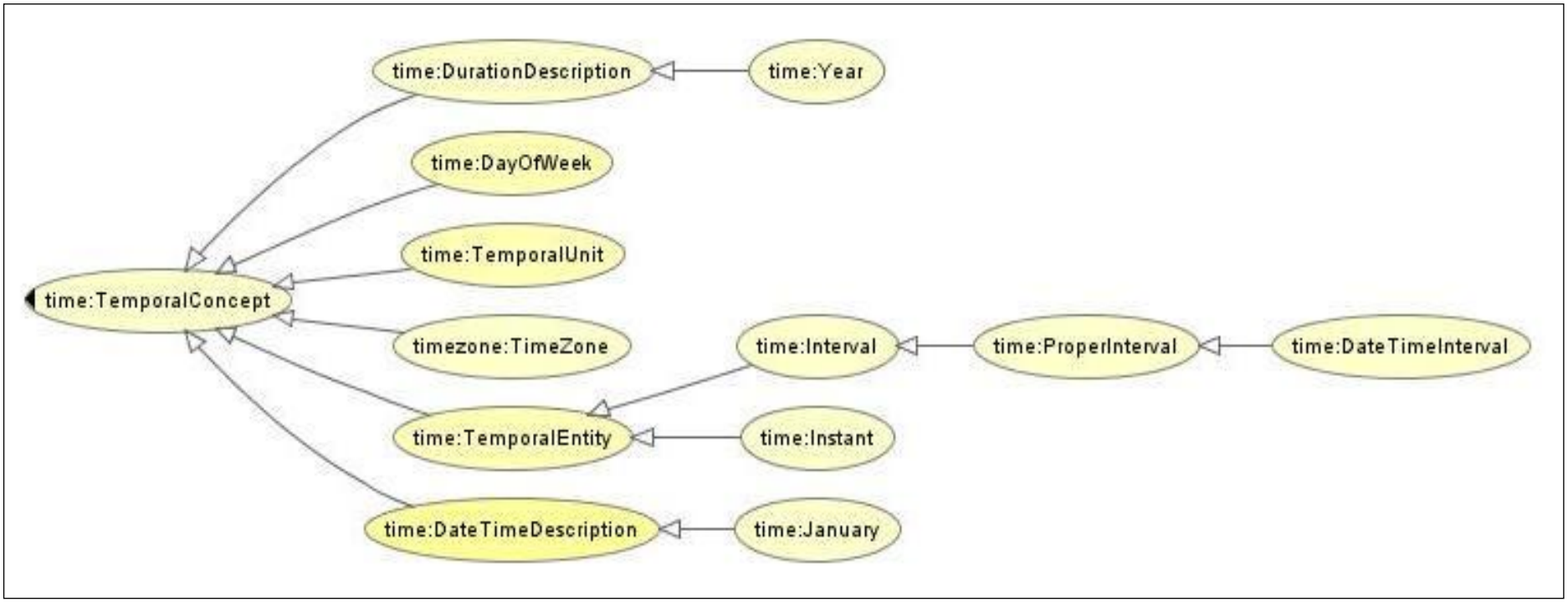

Ilustración 18

6.2.3 ONTOLOGÍA TREND

La ontología trend (o TREN) ha sido creada para describir de forma simbólica el contenido relativo a la visión por computador en una aplicación de seguimiento.

Como puede verse en el diagrama de la página siguiente, existen muchos conceptos incluidos en la ontología tratados durante esta memoria en las secciones de seguimiento. Ejemplos de estos conceptos son frame, blob, track, camera…

Para obtener un nivel de detalle preliminar acerca de esta ontología debe consultarse el punto **3.3.3 Las ontologías en el modelo de conocimiento.**

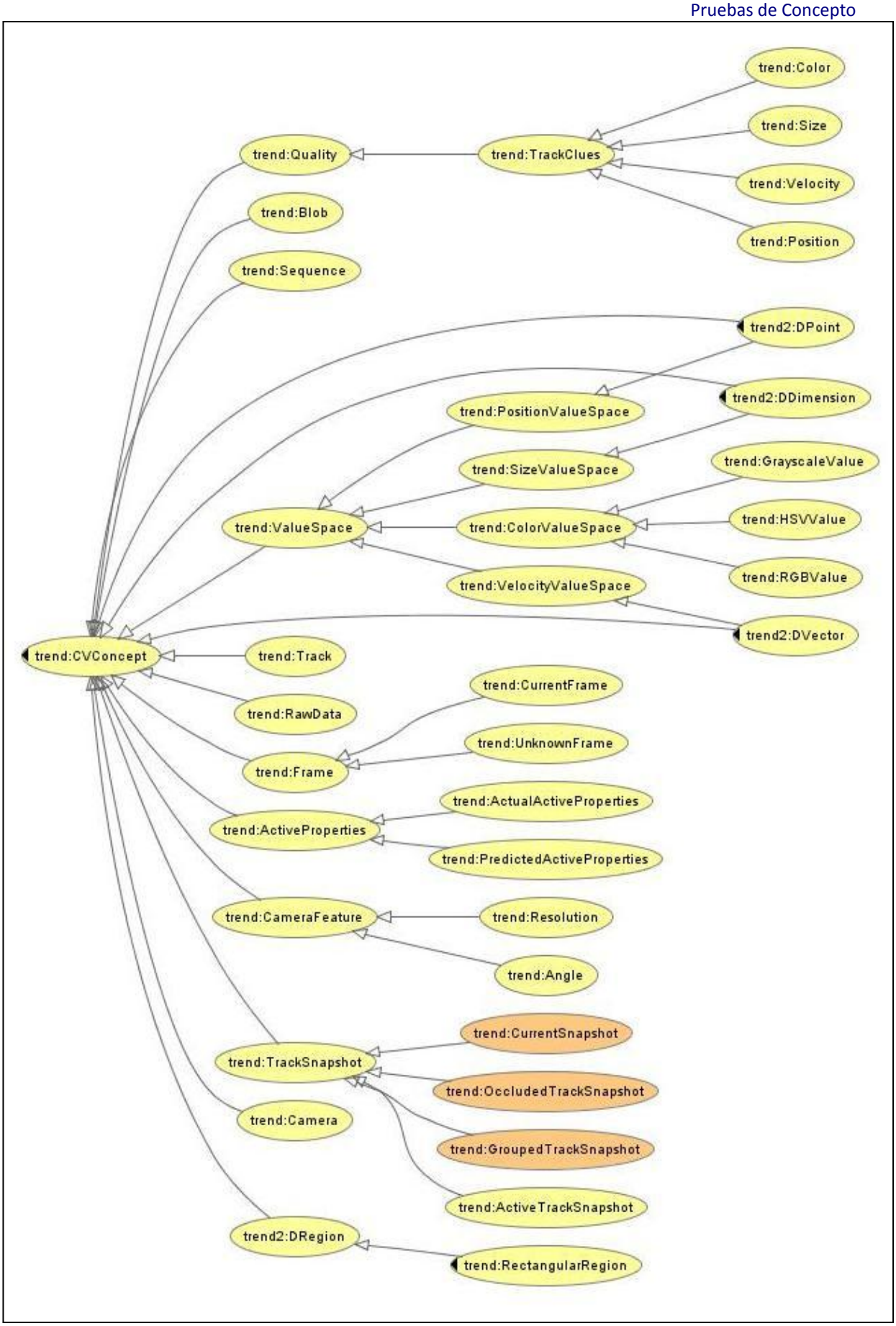

Ilustración 19

6.3 ANÁLISIS DE RESULTADOS

Para demostrar el funcionamiento del enlace entre capas basta con un simple ejemplo que manifieste la transformación de los elementos numéricos de la capa de seguimiento a los elementos simbólicos de la capa de contexto.

Los tracks o pistas que debe detectar el sistema de seguimiento son pasados a la capa de contexto y representados de forma simbólica, deben tener asociadas una serie de propiedades que se extiendan a lo largo de varios frames.

Estas propiedades deben corresponderse con su posición (x,y), su tamaño, tanto la altura (h) como la anchura (w), etc.

Otra de las cuestiones que van a observarse es la existencia de tantas entidades correspondientes a frames en el modelo de contexto como fotogramas haya en el vídeo. Esto puede garantizarse a través de un contador de fotogramas en el sistema de seguimiento insertado de forma puntual para llevar a cabo las pruebas y cuya salida está representada en la línea de comandos que acompaña la ejecución del vídeo.

En general el análisis de resultados trata de demostrar que es necesario instanciar una serie de ontologías para conformar el modelo simbólico.

6.3.1 INTRODUCCIÓN A JAMBALAYA

A pesar de que Jambalaya es una herramienta simple es necesario realizar una breve introducción de cómo se representan los elementos de una o varias ontologías en este entorno.

Los elementos tales como las clases o los individuos se representan de distintas formas esta es la tabla que indica cada una de las representaciones.

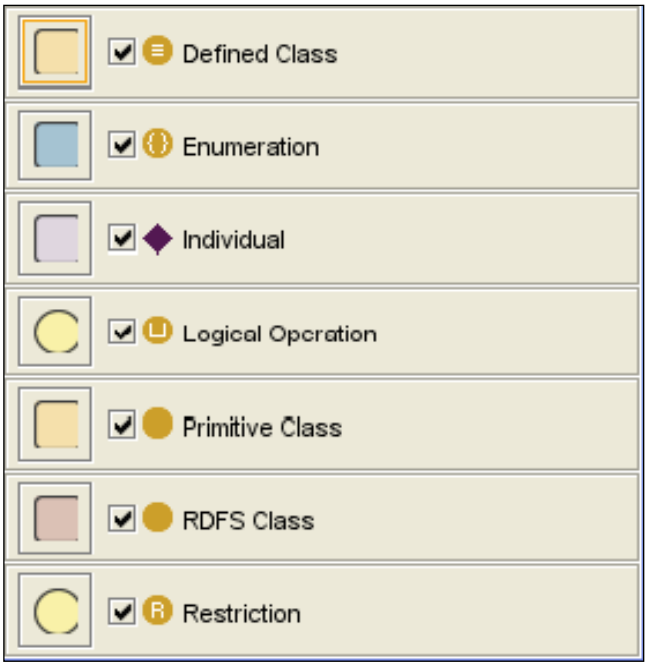

Ilustración 20

En general es fácil entender cuál es el cometido de cada elemento, únicamente el caso de RDFS class puede resultar más complejo.

RDFS class permite declarar recursos como clases para otros recursos. Para el caso particular que se va a ver en el siguiente punto se utiliza para declarar el recurso OWL-Time como clase para otro recurso, el de la ontología TIme.

Por otro lado, es necesario representar las propiedades tanto de las clases como de las instancias, para hacerlo se utilizan arcos de diferentes colores según la propiedad usada.

Los tipos de propiedad que existen en la aplicación pueden observarse en esta tabla:

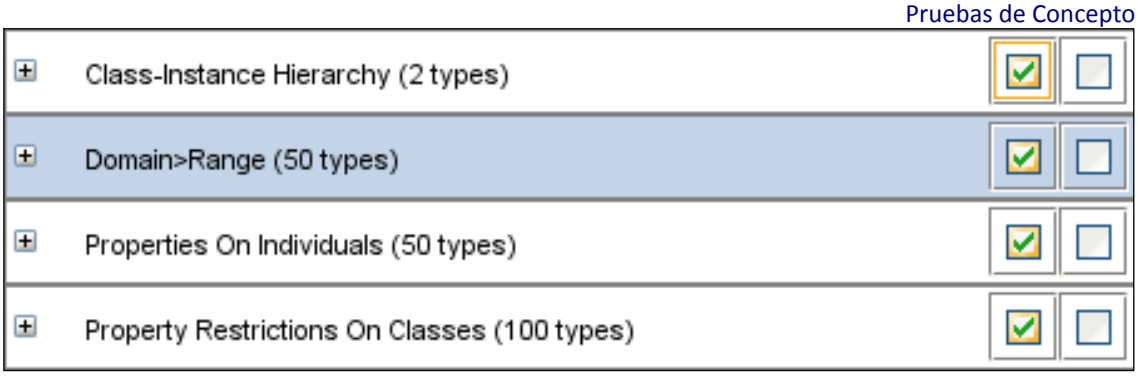

Ilustración 21

Cada tipo de propiedad alberga muchas propiedades lo que desemboca en una gran variedad de arcos de diferentes colores que las representan. Como puede verse en la imagen existen multitud de propiedades de dominio y rango.

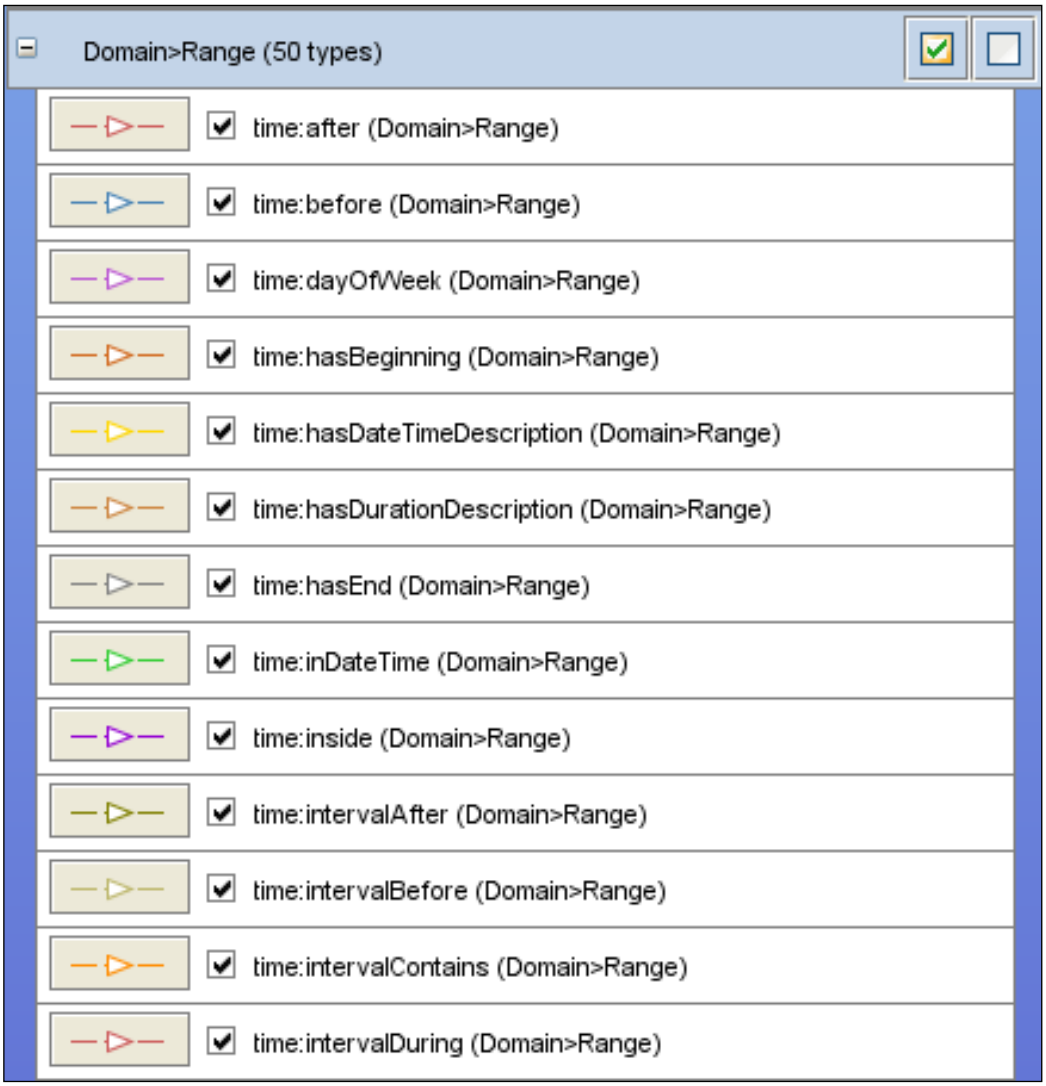

Ilustración 22

Conocer estos elementos de Jambalaya es fundamental a la hora de traducirlos al modelo simbólico generado a partir de una ejecución de la aplicación.

6.3.2 EL MODELO SIMBÓLICO

En esta representación de los resultados van a verse de forma conjunta todos los elementos generados en el modelo, desde las clases a las restricciones pasando por las propiedades y los individuos.

La ontología owl cuyo concepto principal es Thing abarca estas 4 clases.

- Geometry
- Time
- Trend
- OWL-Time

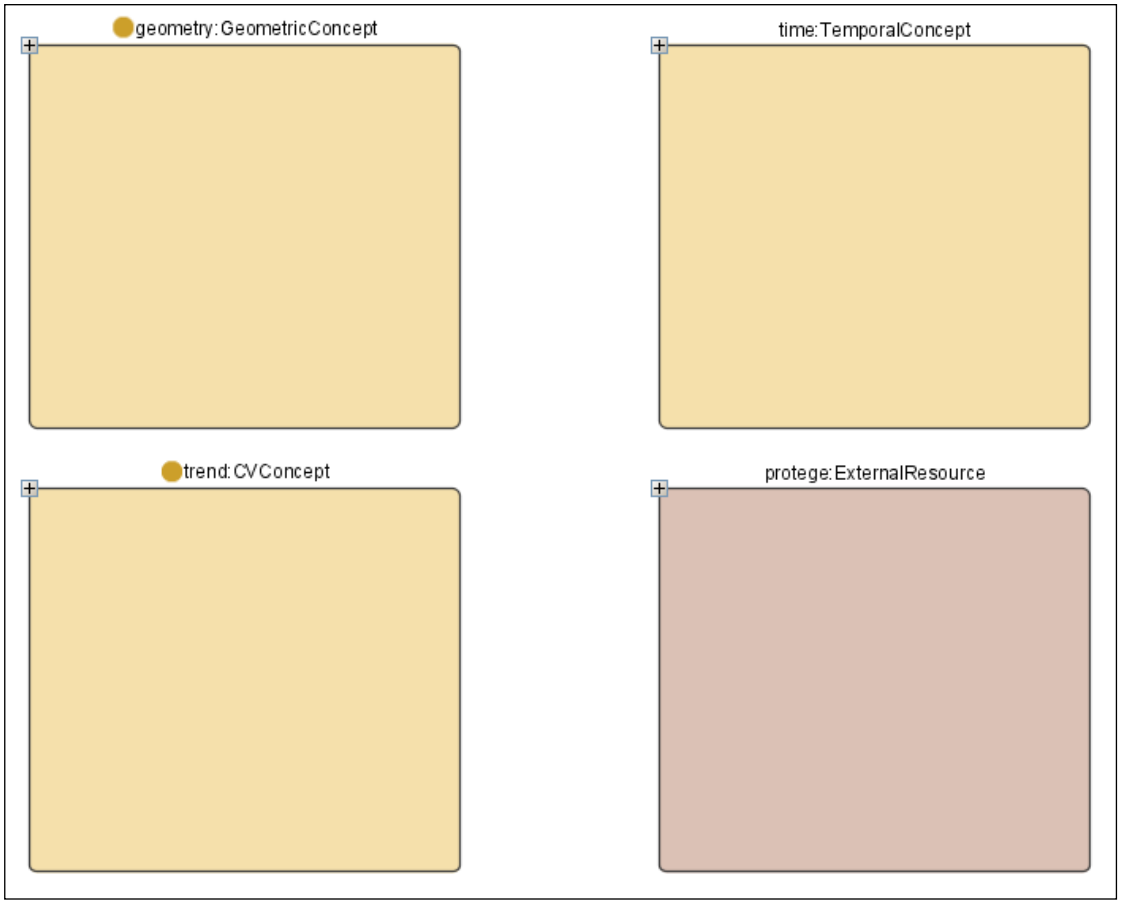

Ilustración 23

El objetivo es centrarse en la ontología trend o TREN que es la que lleva a cabo la representación simbólica de los tracks y sus características.

La imagen siguiente representa las clases, las relaciones y los individuos, contenidos en la clase TREN y generados a partir del vídeo de ejemplo. Todas estas clases según la tabla de elementos vista en **6.3.1 Introducción a Jambalaya** son de tipo primitivo.

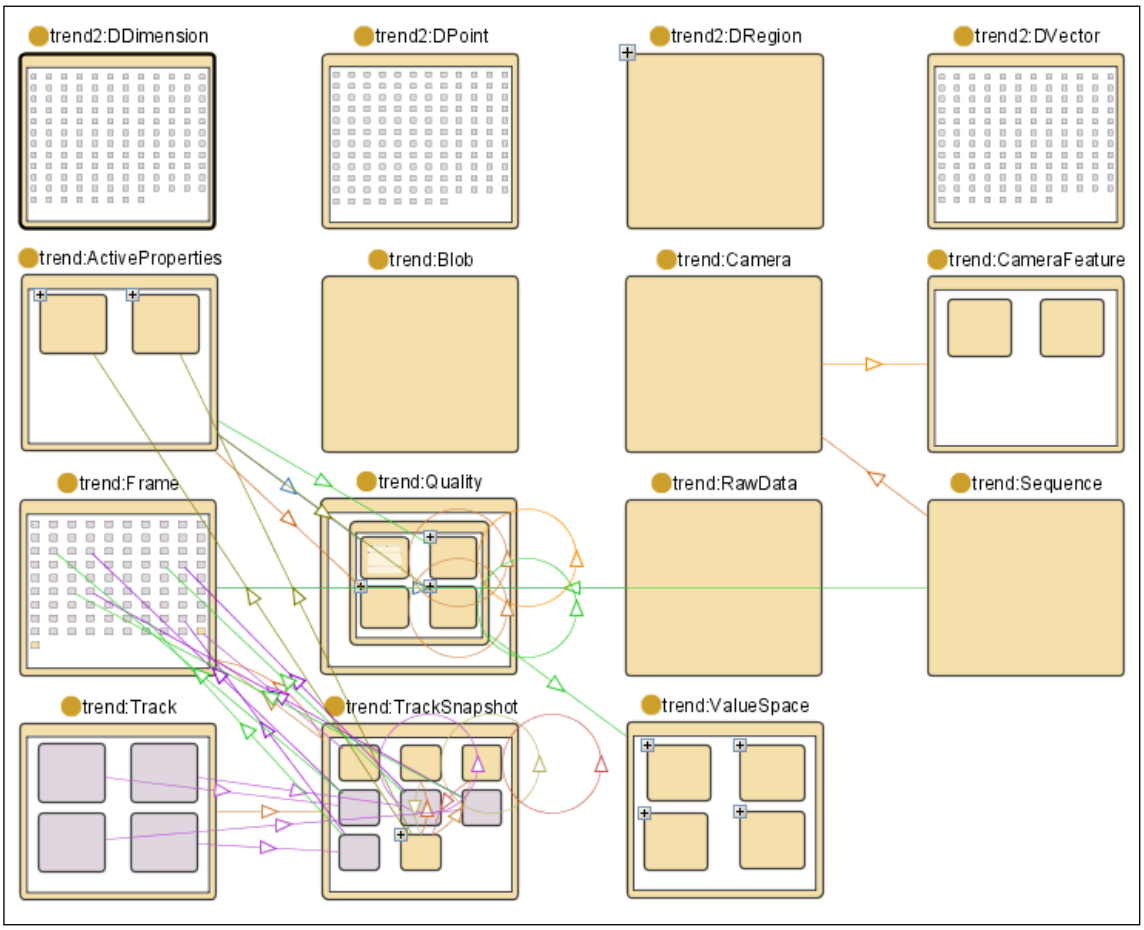

Ilustración 24

Las clases que presentan una mayor cantidad de relaciones son:

- Frame
- Track
- TrackSnapshot
- ActiveProperties
- Quality
- ValueSpace

Esto se debe a que son las clases que despliegan una mayor capacidad de expresividad en el modelo actual para la información obtenida a partir de la capa de seguimiento. El modelo actual de representación simbólica, se encuentra en estas clases y en sus instancias.

La clase Frame

Frame es una clase que trata de dar un representación ontológica en visión por computador para cada frame de un vídeo.

Para el caso del vídeo de ejemplo, esta clase se ha instanciado hasta en 89 ocasiones, debido a que el vídeo contiene 89 fotogramas.

Dentro de la clase Frame existe otra clase, denominada UnknownFrame, con una instancia creada. Cuando se desea asociar cualquier TrackSnapshot a un Frame que indica su finalización y que se desconoce durante el análisis, debe asociarse a un elemento de tipo UnknownFrame. Al finalizar el análisis la finalización de este TrackSnapshot debería asociarse al último Frame del vídeo quedando la instancia de UnknownFrame sin relaciones con instancias de cualquier tipo

Las propiedades más interesantes de Frame son las que le relacionan con la clase TrackSnapshot:

- IsValidInBegin
- IsValidInEnd

Estas 2 propiedades indican que una o varias características de un Track (velocidad, posición, ...) son válidas durante los Frames a los que apuntan. IsValidInBegin indica a partir de que Frame son válidas las características (TrackSnapshot) de un Track. IsValidInEnd indica a partir de que Frame dejan de ser válidas las características de un Track.

La siguiente ilustración muestra todas las instancias creadas de la clase Frame. También pueden observarse las relaciones (líneas de color verde y morado) que representan las propiedades mencionadas y que provienen de la clase TrackSnapshot. La línea verde destacada de la imagen se corresponde con una relación de tipo IsValidInBegin sobre la instancia número 29 de la clase frame.

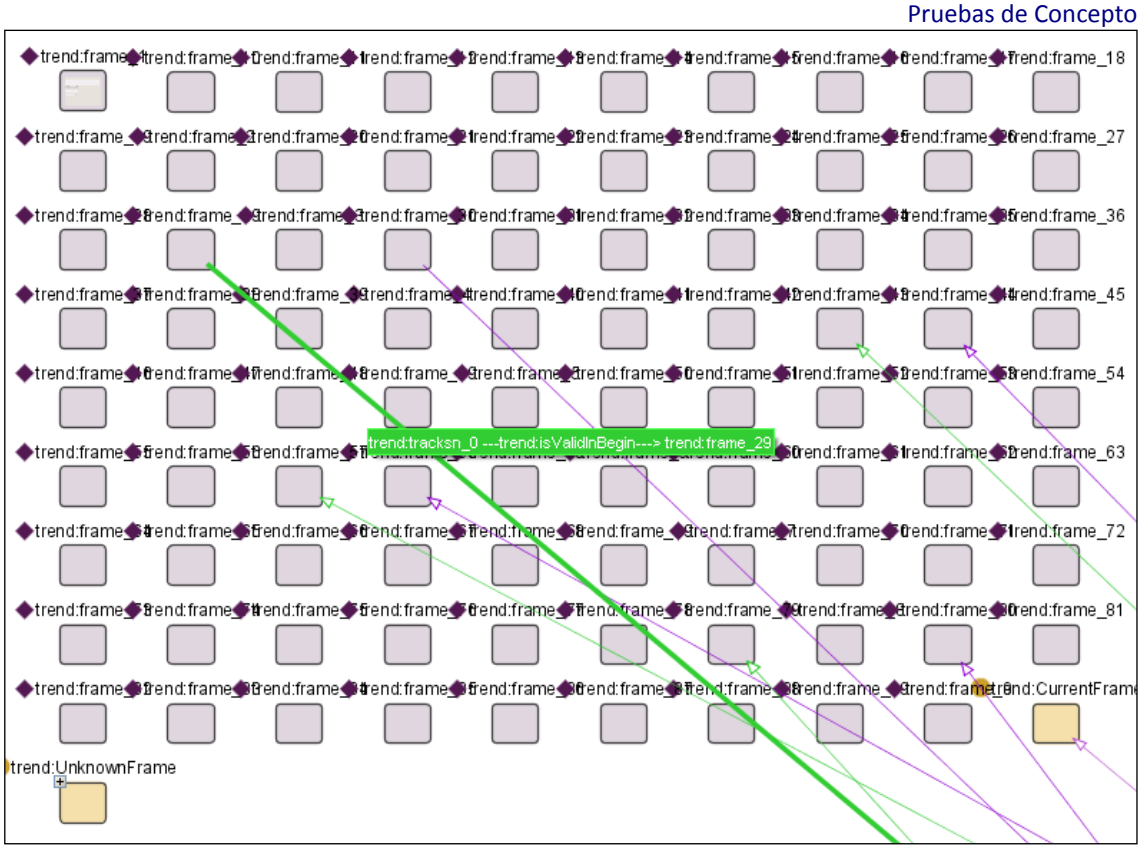

Ilustración 25

A pesar de tener una menor importancia (debido a que no existen instancias para el modelo ejecutado en este caso), Frame está relacionada también con la clase Sequence debido a que una secuencia son un conjunto de Frames y con la clase RawData que representa que cada Frame tiene información de la imagen en crudo.

En esta imagen se puede observar como en la primera instancia de Frame creada, la relación hasRawData queda vacía ya que actualmente no se utilizan.

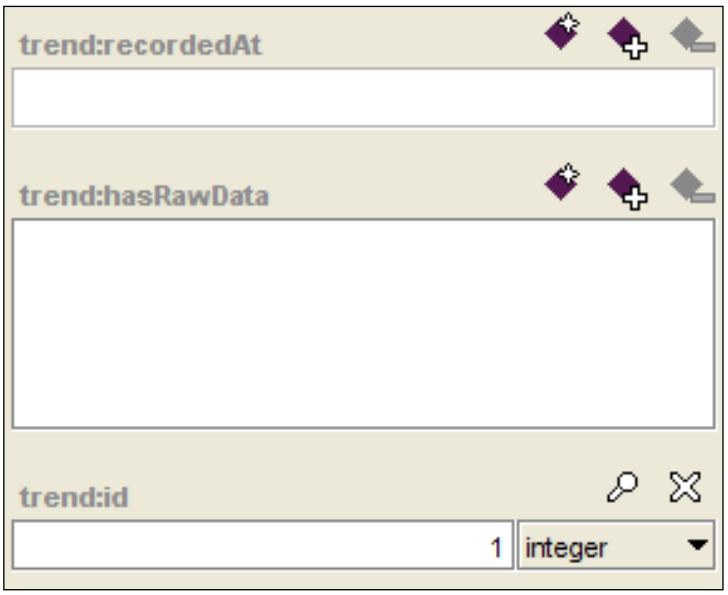

Ilustración 26

La propiedad recorderedAt tiene que ver con la ontología Time y representaría el momento en el que se crearía la instancia de Frame aunque actualmente tampoco se utiliza.

La clase Track

Representa las pistas halladas durante el análisis del sistema de seguimiento.

En este caso se han instanciado 4 tracks, cada instancia es una pista independiente y simbolizan cada avión aparecido en el vídeo.

La propiedad fundamental de la clase Track es hasSnapshot. La propiedad hasSnapshot indica que cada track puede tener varios conjuntos de características.

En la ilustración se puede observar como está destacada la relación entre track_0 y tracksn_0. La traducción de esto en el modelo indicaría que track_0 tiene un conjunto de características guardadas en la instancia de TrackSnapshot, tracksn_0.

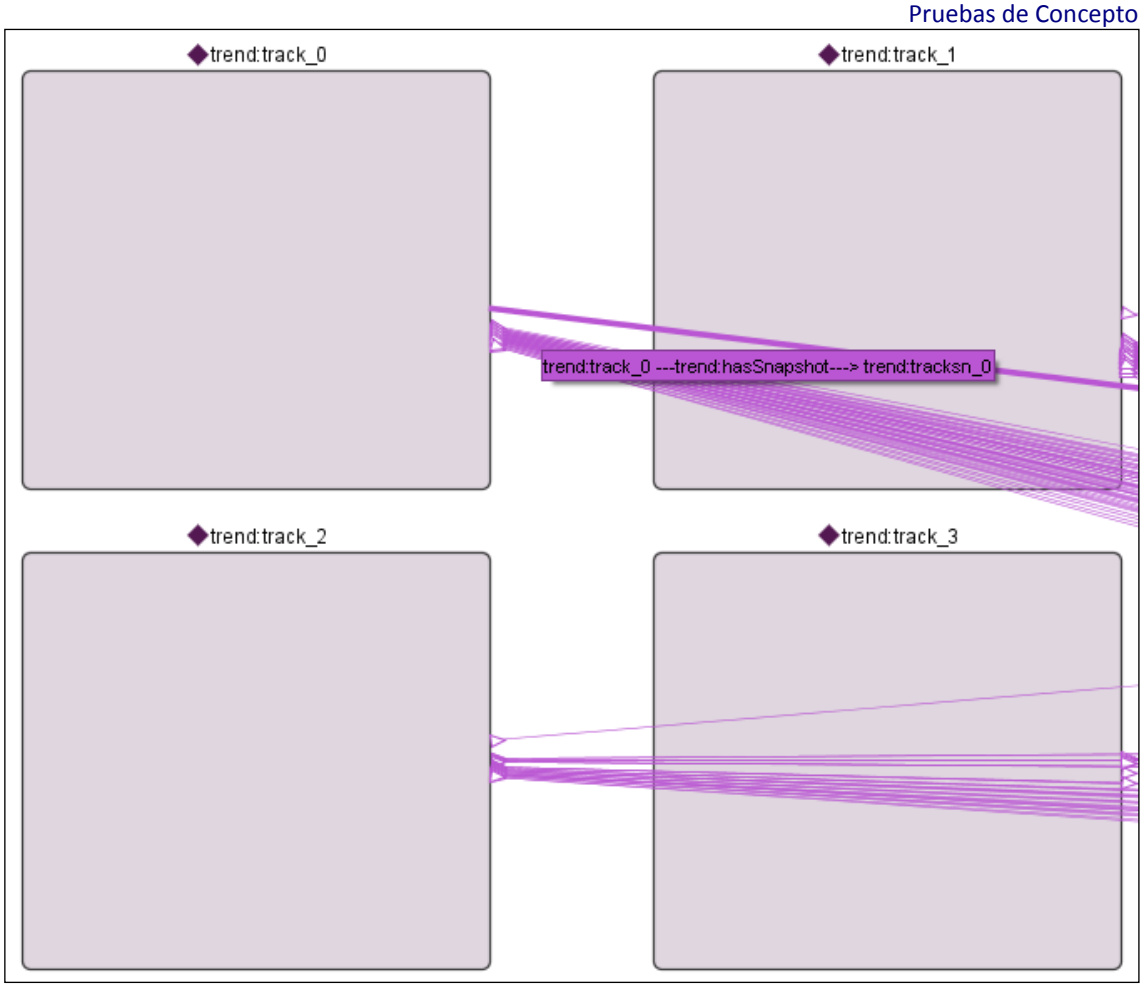

Ilustración 27

La imagen siguiente demuestra que haciendo zoom en track_0 pueden observarse varios conjuntos de características (TrackSnapshot). La razón de este hecho es que las pistas o tracks de un vídeo van cambiando y solo mantienen sus características durante un periodo determinado de frames.

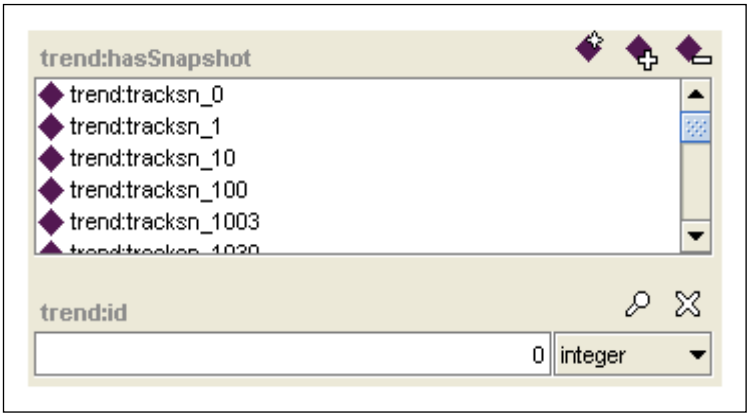

Ilustración 28

La clase TrackSnapshot

Como ya se ha introducido anteriormente TrackSnapshot representa un conjunto de características de un único Track, que solo es válido durante un periodo determinado de Frames, pero es solo su subclase ActiveTrackSnapshot la que en realidad contiene las instancias que llevan a cabo esta función.

Esta ilustración muestra el contenido de la clase TrackSnapshot, en la que existen 4 clases y 4 instancias.

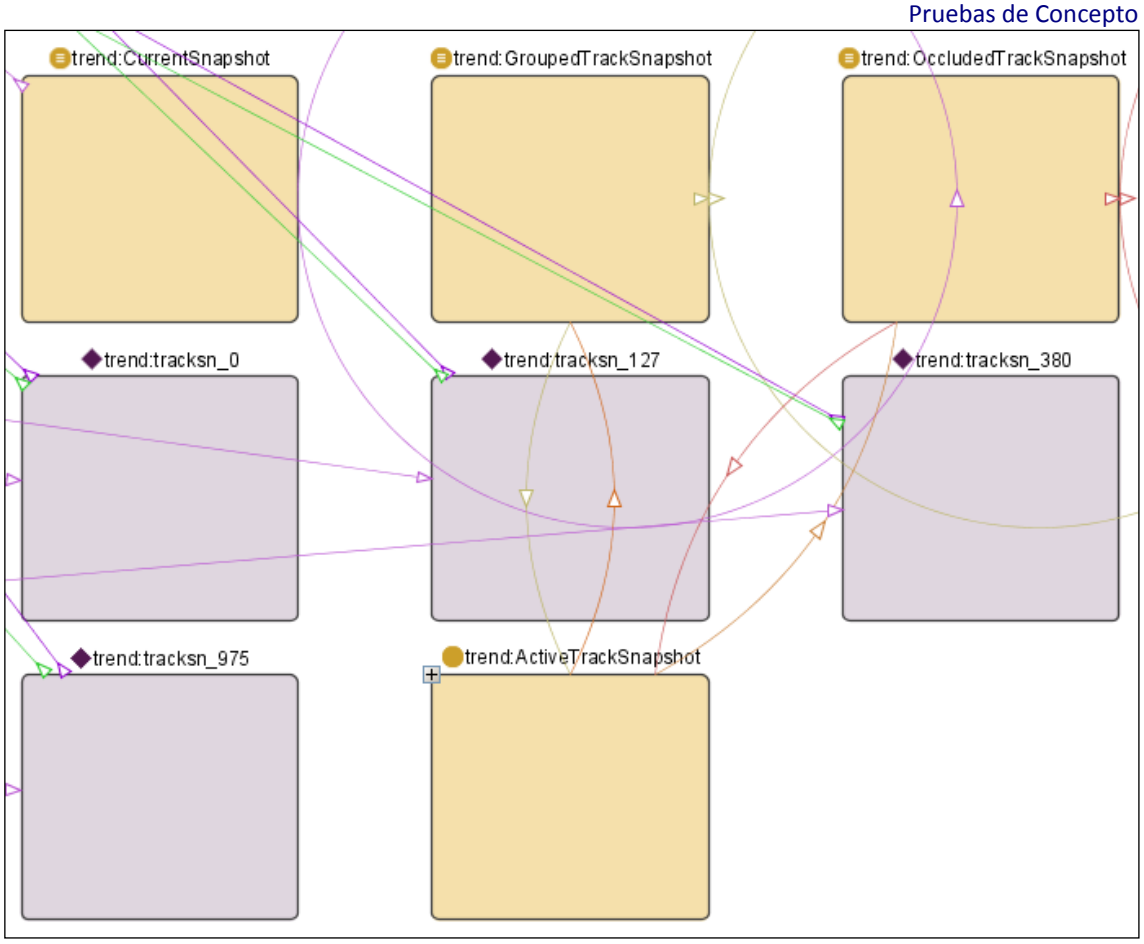

Ilustración 29

Las instancias que se observan en TrackSnapshot y que no están incluidas en la clase ActiveTrackSnapshot, no contienen características acerca del Track por el que son apuntadas.

La clase TrackSnapshot contiene las clases:

- CurrentSnapshot: Indica un Snapshot relacionado con un CurrentFrame, estas relaciones solo se utilizan durante el análisis, cuando este termina no existen elementos Current de ningún tipo.
- GroupedTrackSnapshot: Será utilizada en el futuro, mediante la relación isGroupedBy, para indicar a través de ActiveTrackSnapShot cuando un Track esta agrupado a otro.
- OccludedTrackSnapshot: Será utilizada en el futuro, mediante la relación isOccludedBy, para indicar a través de ActiveTrackSnapShot cuando un Track ocluye a otro.

- ActiveTrackSnapshot: Esta clase es la que contiene las instancias que referencian a los Tacks y sus características, denominadas ActualActivePropierties y PredictedActiveProperties, que pertenecen a la clase ActiveProperties.

Haciendo zoom sobre una instancia de ActiveTrackSnapshot se puede advertir como contiene una referencia a las características, tanto actuales como futuras, de un Track, y una referencia a los Frames durante los cuales estas características son válidas.

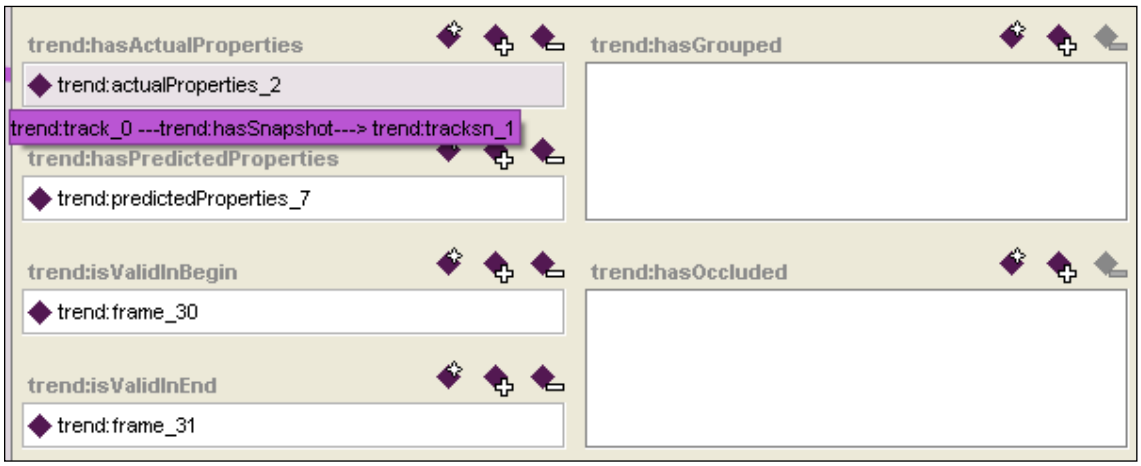

Ilustración 30

La clase ActivePropierties

ActivePropierties representan un conjunto de características de una pista cuando esta permanece en escena, es decir, cuando es considerado un track activo. Debido a la posibilidad de que las pistas tengan características actuales y futuras o previstas, esta clase tiene en realidad dos subclases que son las que se instancian:

- ActualActivePropierties: Representa las características actuales de un Track.
- PredictedActivePropierties: Representa las características previstas para un Track.

ActiveProperties está relacionada con ActiveTrackSnapshot de forma que cada instancia de la segunda contiene referencias a instancias de los 2 subtipos de la primera. Las propiedades que lo permiten se denominan hasActualProperties y hasPredictedProperties.

En esta ilustración puede observarse como la relación hasActualProperties destacada en verde que proviene de ActiveTrackSnapshot apunta a la clase ActualActiveProperties.

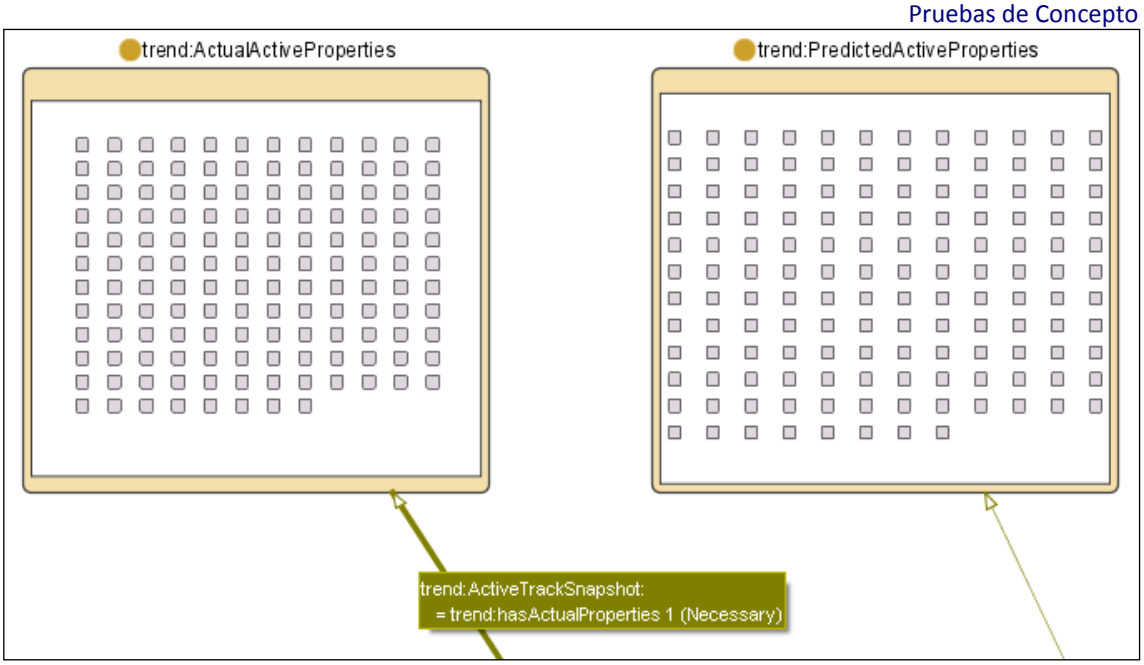

Ilustración 31

Haciendo zoom sobre una de las instancias de la clase ActualActiveProperties puede observarse que es capaz de albergar las características de un Track:

- Color
- Posición
- Tamaño
- Velocidad

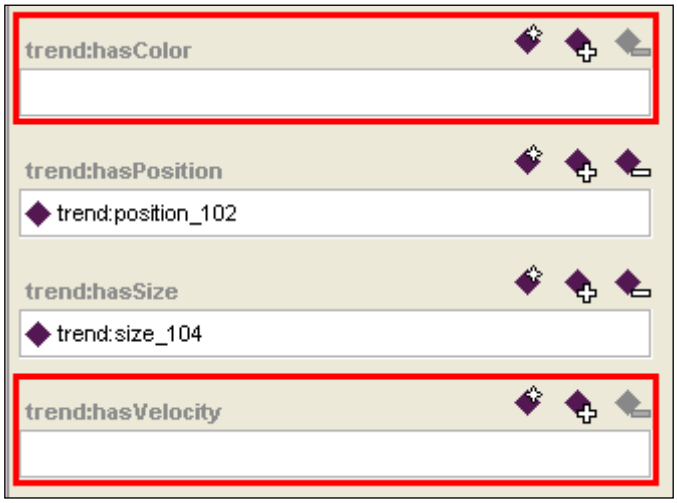

Ilustración 32

Actualmente las características de color no son transmitidas al sistema de contexto a través del sistema de seguimiento por lo que no se ven representadas.

Las características de velocidad son usadas en las instancias de la clase PredictedActiveProperties.

Las instancias de todas estas características son albergadas por la clase Quality y se relacionan con estas características activas a partir de las propiedades:

- hasColor
- hasPosition
- hasSize
- hasVelocity

La clase Quality

Quality tiene como subclase TrackClues, esta subclase mantiene una relación con la clase ValueSpace, que a su vez heredan todas sus subclases, que se denomina hasValue. Esta relación además de unir las subclases de TrackClues y ValueSpace, dando así valores concretos a las características de los Tracks, une las ontologías Trend y Geometry debido a que las subclases de ValueSpace pertenecen a esta segunda.

Las subclases de TrackClues son:

- Color
- Position
- Size

- Velocity

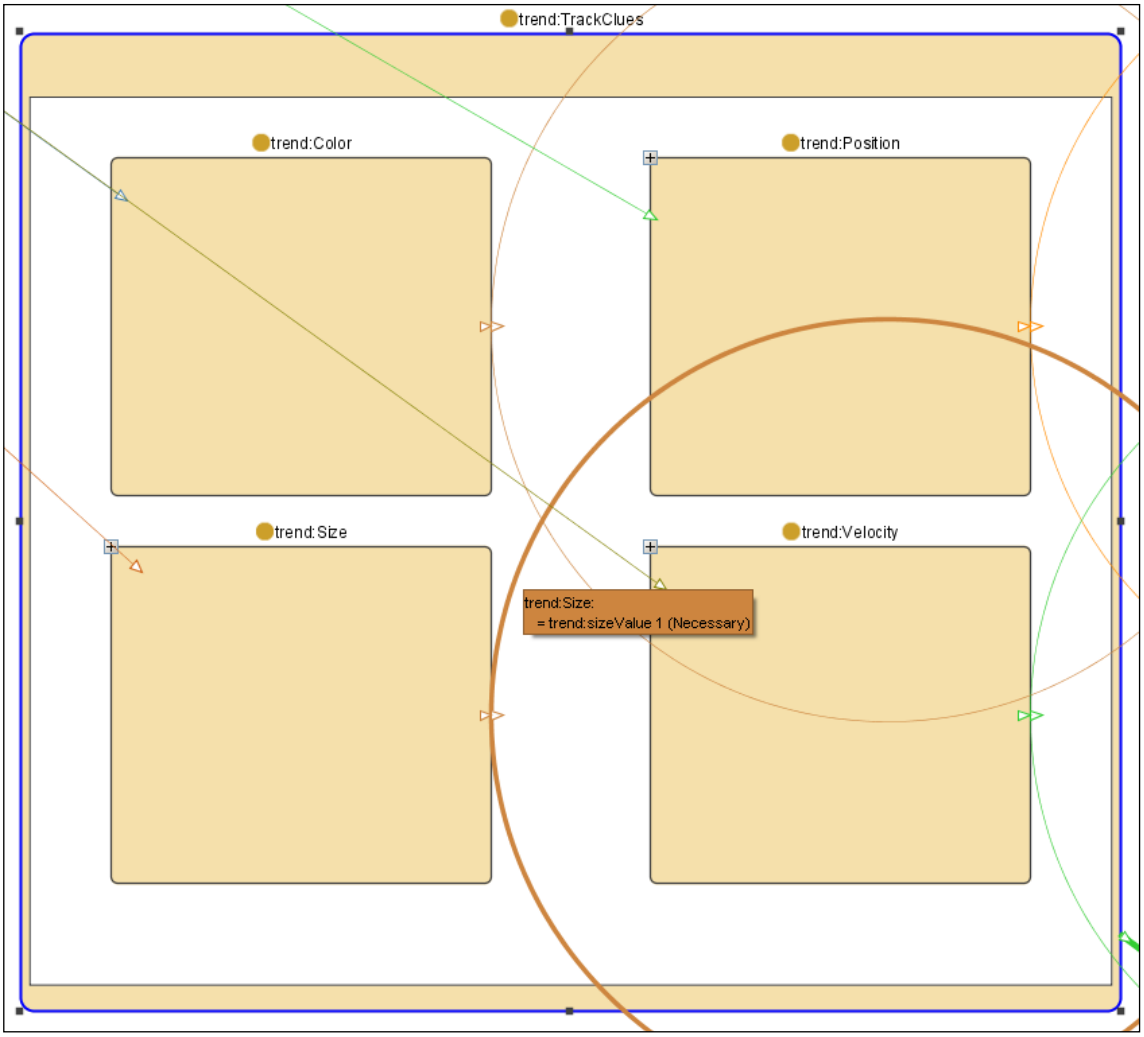

Ilustración 33

Estas subclases son las que se instancian para dar valores a las características de los tracks. Existe una relación que se autoreferencia en cada una de estas clases, su significado es que, para cada instancia de cada clase existe un único elemento que define los valores correspondientes a la característica que representa la clase, estas relaciones se denominan:

- colorValue
- positionValue
- sizeValue
- velocityValue

Esta imagen muestra la clase Size con todas sus instancias, las dos relaciones que se observan son en realidad la misma relación.

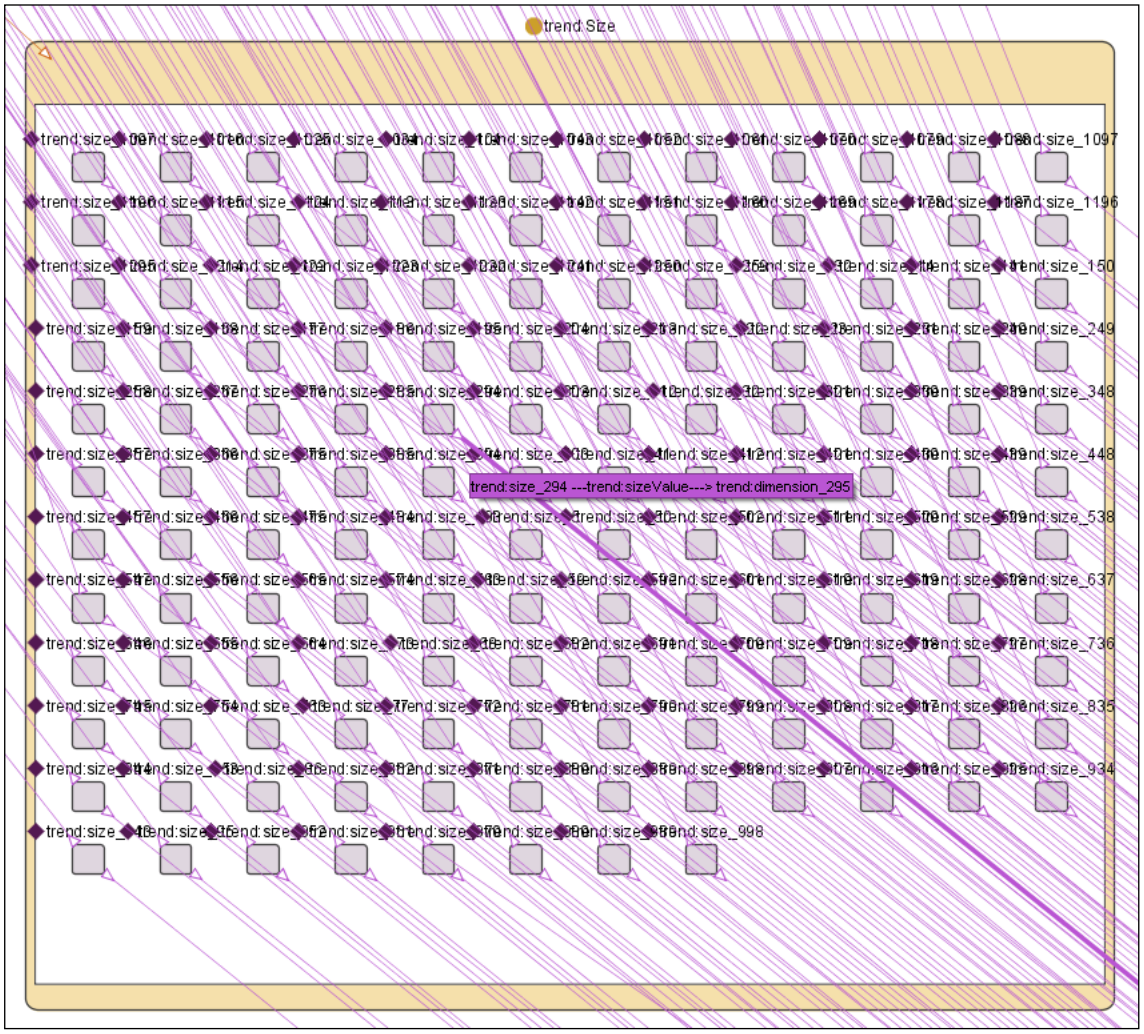

Ilustración 34

Una apunta a la subclase correspondiente en ValueSpace, llamada DDimension, que es contenida por SizeValueSpace.

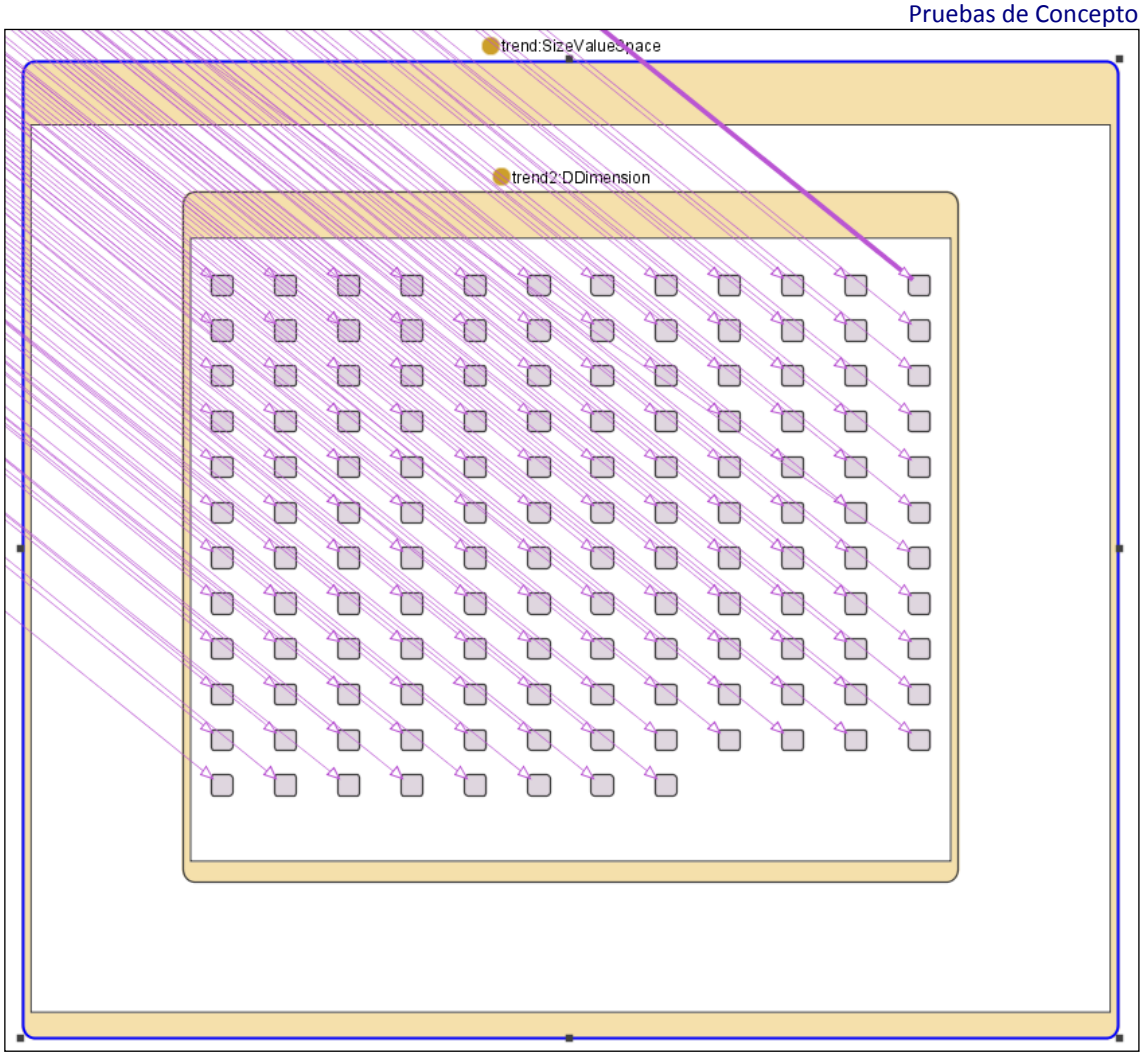

Ilustración 35

La otra apunta directamente a la clase misma como elemento perteneciente a la ontología Geometry.

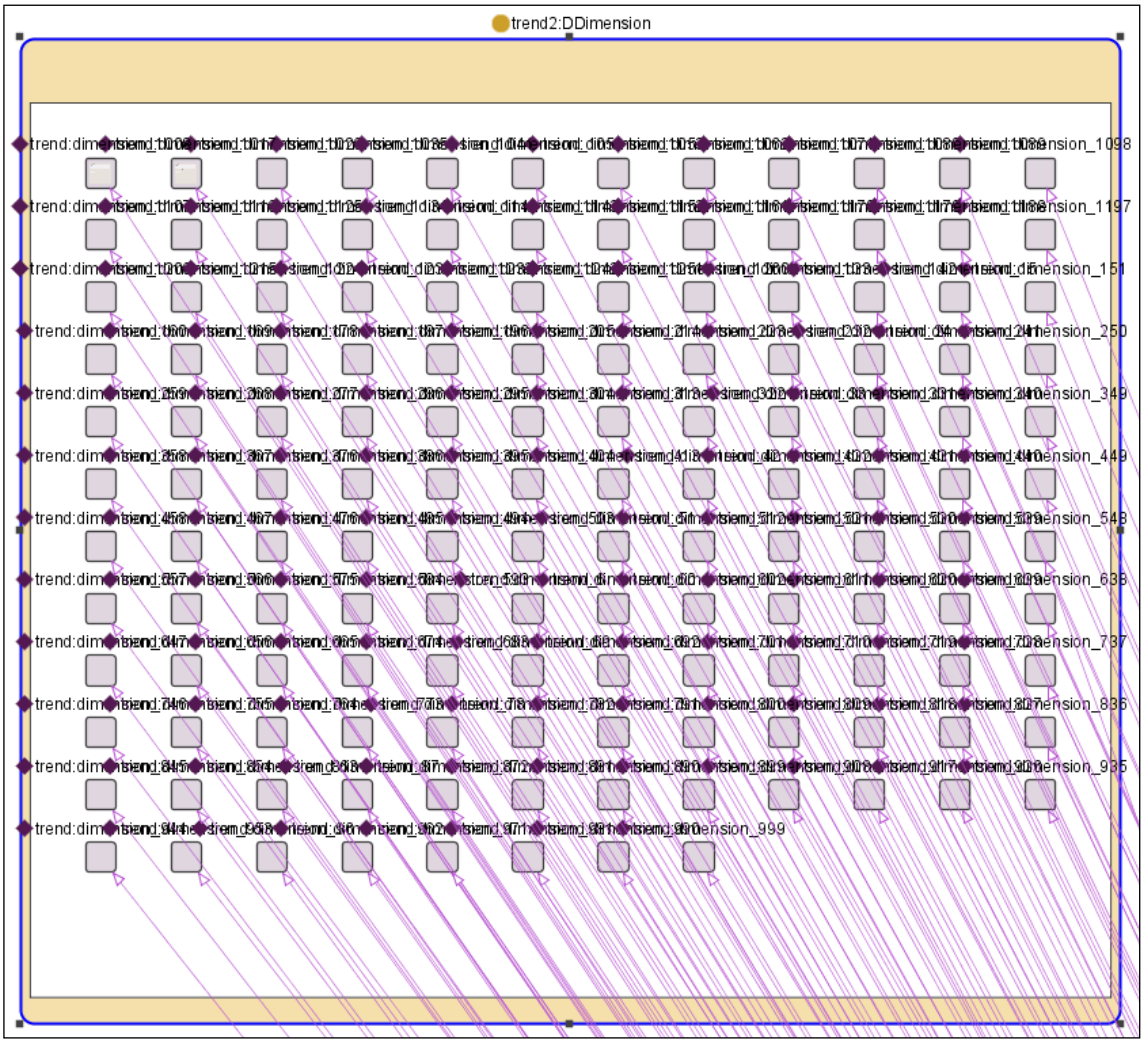

Ilustración 36

La clase ValueSpace

La clase ValueSpace trata de representar los espacios de valores de las diferentes características de los Tracks.

Del mismo modo que la clase TrackClues contiene cuatro clases para definir cada uno de los tipos de características de los Tracks, ValueSpace contiene cuatro clases para definir los espacios de valores de estas características:

- ColorValueSpace
- PositionValueSpace

- SizeValueSpace
- VelocityValueSpace

Ilustración 37

Actualmente los espacios de valores pueden contener varias formas de expresar una característica, en el caso del color por ejemplo, ya se han proyectado 3 subclases que se corresponderían con 3 formas de expresar el color:

- GrayScaleValue
- HSVValue
- RGBValue

Para cada una de las demás clases, hoy por hoy, solo existe una representación.

- DPoint para la clase PositionValueSpace, que representa la posición de un Track.

- DDimension para la clase SizeValueSpace, que representa el tamaño de un Track.
- DVector para la clase VelocityValueSpace, que representa la velocidad de un Track.

Las instancias de estas clases, pertenecientes tanto a Geometry como a Trend, son las que contienen los datos en bruto de las características. En la siguiente ilustración se muestra el contenido de una instancia de la clase DPoint que a través de los valores describe una posición determinada.

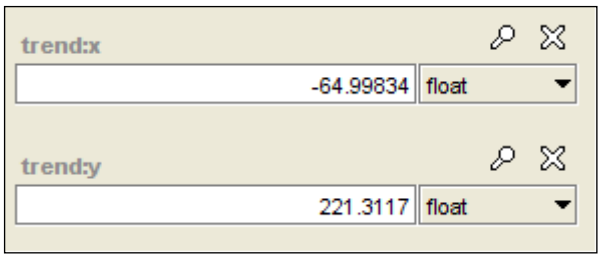

Ilustración 38

Para finalizar se muestra un diagrama general de las relaciones de cada instancia derivadas de las características de los Tracks. En él se observa la correspondencia entre las propiedades actuales y previstas y sus valores finales en instancias de clases que describen un conocimiento de bajo nivel.

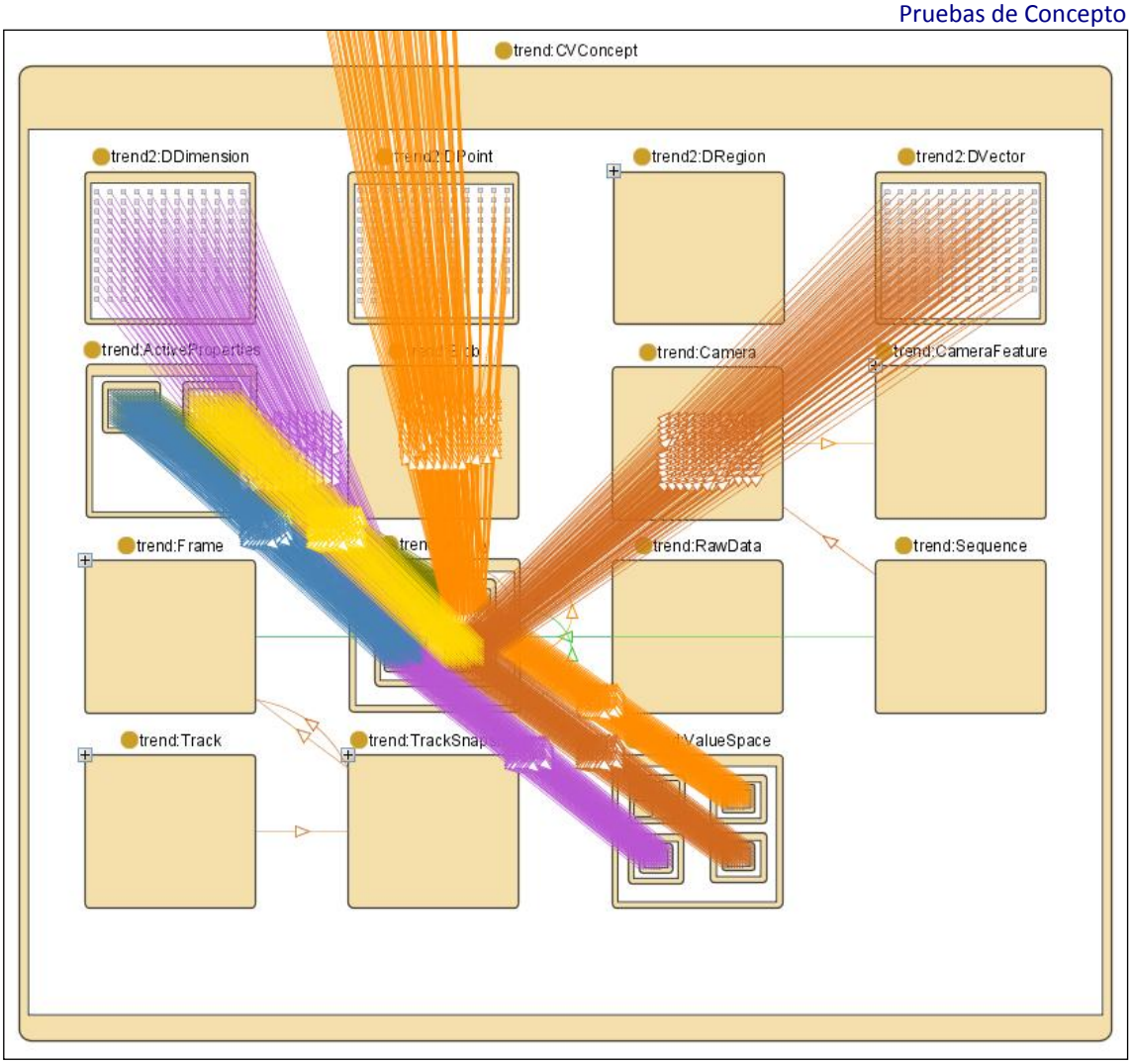

Ilustración 39

6.4 PRUEBAS DE INCREMENTO TEMPORAL

Las pruebas sobre el incremento del tiempo se realizan para conocer el impacto que ha tenido sobre el sistema de seguimiento la inclusión de la capa de contexto basada en ontologías.

Desde un primer momento se sabía que este añadido iba a deteriorar el rendimiento del sistema debido al gasto en recursos de la máquina virtual, las llamadas a un código que debe representar la información pasada…

Estas pruebas sirven para hacerse una idea preliminar de la pérdida de rendimiento que se produce al ensamblar un sistema de ontologías desarrollado en Java a un sistema de seguimiento desarrollado en C++.

6.4.1 EL SISTEMA DE PRUEBAS

El sistema utilizado para llevar a cabo las pruebas tiene las siguientes características:

- Procesador: Intel Core 2 4400 a 2.00 GHz.
- Memoria RAM: 1 GB.
- Sistema Operativo: Microsoft Windows XP Profesional. Service Pack 3.

Teóricamente este sistema ve afectado en mayor medida su rendimiento según el tipo de procesador utilizado. Un salto de gama en el procesador puede provocar mejorar sustanciales en el tiempo de análisis.

6.4.2 CARACTERÍSTICAS A TENER EN CUENTA

Las pruebas se han realizado con varios vídeos de características diferentes para observar los motivos más graves de la pérdida de rendimiento del sistema con la inclusión de la capa de contexto.

Las características a tener en cuenta en el análisis de los videos son:

- El número de pistas que aparecen en el vídeo de forma simultánea. Un número grande de pistas debería devaluar los tiempos debido a que se hace un uso intensivo de la interfaz de comunicación enviando las ordenes de actualización, creación y borrado de estas pistas.
- El número de fotogramas del vídeo. Este parámetro es importante por dos razones, en primer lugar porque el análisis se lleva a cabo para cada frame del vídeo y no para cada segundo del vídeo, el número de fotogramas es el número de ciclos que realiza la

ejecución. En segundo lugar porque una parte importante del análisis es conocer cuánto tiempo se tarda en analizar cada frame.

Se han realizado tres pruebas de tres videos cada una, sobre entornos distintos con un número de pistas diferente. Los videos denominados PruebaPasilloX tienen por norma general 0, 1 o 2 pistas, la mayoría del tiempo solo debería detectarse una pista. Los videos llamados PruebaCalleX tienen entre 1 y 5 pistas, la mayoría del tiempo se detecta 3 o 4 pistas. Los videos PruebaFutbolX tienen entre 2 y 10 pistas y la mayoría del tiempo se detecta entre 6 y 8 pistas.

En esta tabla pueden observarse los datos relativos a los fotogramas de cada vídeo.

Título	Fotogramas/segundo	Tiempo total	Fotogramas
PruebaPasillo1	25	20	500
PruebaPasillo2	25	20	500
PruebaPasillo3	25	20	500
PruebaCalle1	10	20	200
PruebaCalle2	10	20	200
PruebaCalle3	10	20	200
PruebaFutbol1	25	20	500
PruebaFutbol2	25	20	500
PruebaFutbol3	25	20	500

Las características que no se deben tener en cuenta son:

- La resolución del vídeo. Esta no es una característica a tener en cuenta en el análisis de la capa de contexto. Es cierto, que cuanto mayor es el alto y el ancho de la imagen mayor es también el número de pixeles y esto puede afectar al análisis de un sistema de vídeo, pero la capa de contexto no utiliza los pixeles como parte del análisis y por ello la interfaz de comunicación no envía esta información. Este parámetro es útil para una prueba del rendimiento de la capa de seguimiento más que de la capa de contexto.

En esta tabla se observan los datos de ancho y alto de los fotogramas de cada vídeo y el número de píxeles de cada imagen.

Título	Ancho fotograma	Alto fotograma	Píxeles/Fotograma
PruebaPasillo1	768	576	442368
PruebaPasillo2	768	576	442368
PruebaPasillo3	768	576	442368
PruebaCalle1	384	288	110592
PruebaCalle2	384	288	110592
PruebaCalle3	384	288	110592
PruebaFutbol1	720	576	414720
PruebaFutbol2	720	576	414720
PruebaFutbol3	720	576	414720

6.4.3 RESULTADOS DE LAS PRUEBAS

Los datos que se muestran a continuación se refieren al desarrollo actual, pueden empeorar debido al uso del motor de inferencia o al uso de nuevos niveles en el sistema de ontologías. Es necesario recordar que estos datos solo implican el uso de la interfaz de comunicación JNI y el uso de un único nivel de ontologías.

En esta primera imagen puede observarse el número de frames por segundo del sistema de seguimiento antiguo. La cantidad de frames analizados es bastante alta, se observa un mayor ratio de frames por segundo en los videos PruebaCalleX debido a que la resolución es más baja. Esto demuestra lo que se apuntaba anteriormente en el apartado **6.4.2 Características a tener en cuenta** sobre el efecto de la resolución en el sistema de seguimiento.

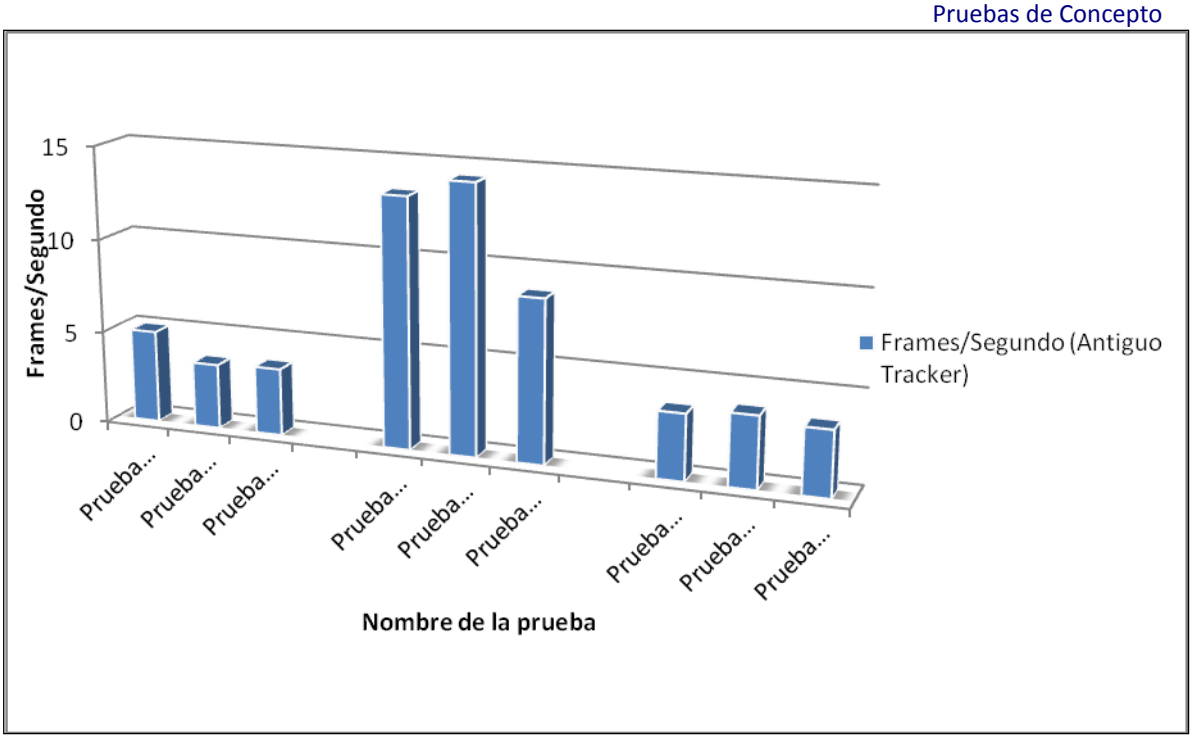

Ilustración 40

Una vez se conoce el elemento que perturba los tiempos en el sistema de seguimiento se desea conocer que elemento perturba los tiempos en el sistema completo.

Como se ve en la tabla los tiempos han empeorado de forma dramática con la inserción de la capa de comunicación y de contexto en el sistema de seguimiento. Los datos parecen mostrar que los tiempos del segundo vídeo son mejores que los del primero, esto es muy chocante ya que la segunda tanda (PruebaCalleX) de videos tiene más pistas que la primera (PruebaPasilloX).

Título	Tiempo tracker antiguo	Tiempo tracker nuevo
PruebaPasillo1	101	1405
PruebaPasillo2	144	2825
PruebaPasillo3	139	1242
PruebaCalle1	15	601
PruebaCalle2	14	1022
PruebaCalle3	23	1117

PruebaFutbol1	142	6878
PruebaFutbol2	131	4597
PruebaFutbol3	143	5368

Si se observa el tiempo invertido por el sistema de seguimiento detenidamente y se comprueba la cantidad de tiempos de las pruebas del tracker antiguo que caben en los tiempos del tracker nuevo se desentraña este misterio. El gráfico siguiente muestra esa información, se observa como la pérdida de rendimiento en comparación con los tiempos del tracker antiguo es mayor en los 2 últimos casos. Aunque de nuevo encontramos una incongruencia en el razonamiento, ahora el ratio es mayor en el segundo caso que en el tercero, sin embargo, se conoce que en la tercera tanda hay en general un número mayor de pistas que analizar. Esto se debe a que los tiempos de análisis del sistema de seguimiento antiguo para la segunda tanda son muy bajos debido a que tanto la resolución de los videos como el número de fotogramas es mucho menor, cuando realizamos la operación de división de los tiempos del sistema nuevo con los tiempos del sistema antiguo, el resultado se dispara debido a que el denominador es muy bajo.

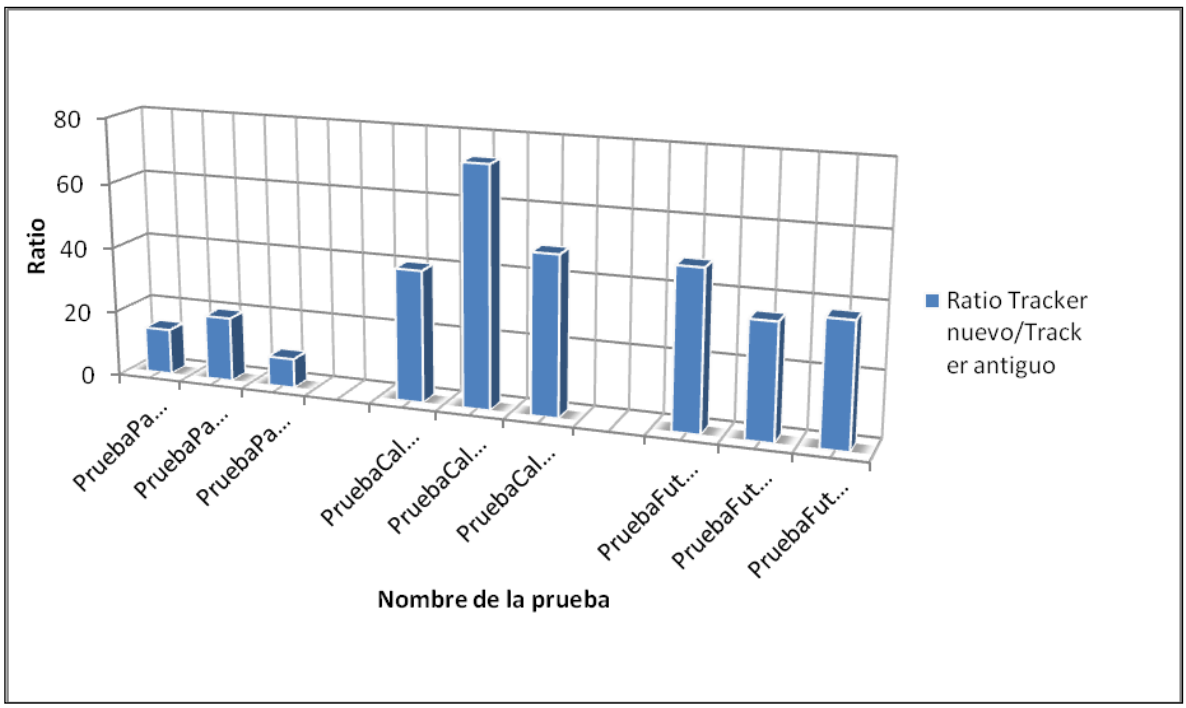

Ilustración 41

La siguiente imagen muestra los segundos consumidos en cada frame para los dos sistemas, el nuevo y el antiguo. En la primera tanda de videos existe un vídeo, el segundo, con un tiempo de análisis por frame similar al de la segunda tanda. Encontramos de nuevo, algo que parece contradecir la idea de que según se incrementa el número de pistas empeora el rendimiento del sistema.

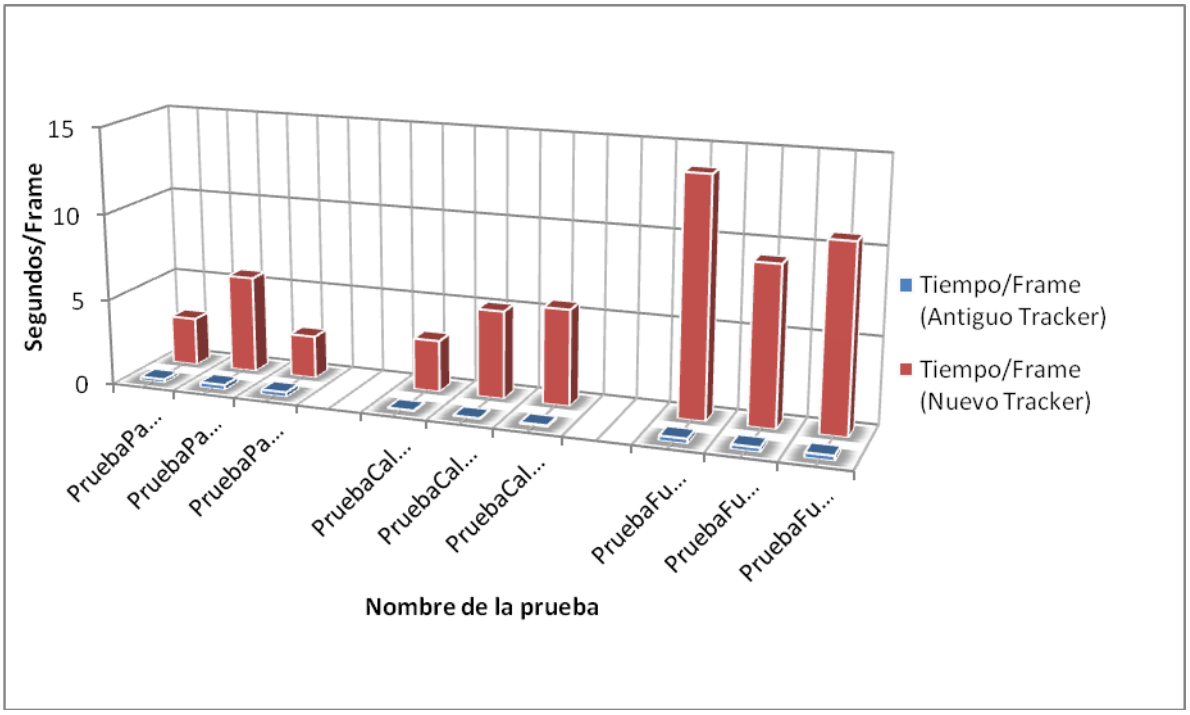

Ilustración 42

Sin embargo, tras observar el analisis de este vídeo detenidamente se advierte que aunque solo deberia existir una única pista, el sistema de seguimiento en su capa de bajo nivel crea tres pistas, lo que se aproxima al número de pistas de la segunda tanda de videos y confirma la teoria del bajo rendimiento cuando existe un uso intensivo de la interfaz de comunicación.

7 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo el lector puede encontrar las conclusiones finales de este proyecto y la dirección que deberían tomar futuros trabajos que amplíen o modifiquen el funcionamiento de la aplicación resultante.

7.1 CONCLUSIONES

Este proyecto ha llevado a cabo la comunicación entre el sistema de seguimiento OpenCV desarrollado en código C++ y un sistema de ontologías escrito en código Java.

A lo largo del desarrollo de este proyecto se ha aprendido el uso de multitud de utilidades descritas en el apartado **1.3.2 Las herramientas utilizadas** y múltiples tecnologías descritas a lo largo de esta memoria:

- Implementación en JNI de una capa de comunicación para unir las capas de seguimiento y contexto.
- Estudio del sistema de seguimiento de OpenCV para desarrollar de forma correcta la capa de comunicación.
- Introducción a la tecnología de contexto basada en ontologías utilizada en la capa de alto nivel.

Se ha llevado a cabo una interfaz gráfica de usuario para gestionar de forma fácil y simple todo este sistema. Para ello ha sido necesario hacer uso de tecnologías como:

- Windows Forms y gestión de eventos para el desarrollo de los formularios de la interfaz.
- Lectura, escritura de XML con herramientas a medida, además del diseño de la estructura de estos documentos, para la carga, guardado y configuración del sistema, lo que aporta propiedades de adaptabilidad. Estas propiedades son muy útiles si en un futuro próximo se desean introducir parámetros referidos a la capa de contexto o ampliar los parámetros del sistema de seguimiento.
- Creación de proyectos de instalación para la integración de las capas y la interfaz en un solo paquete instalable.

En general se consideran cumplidos los objetivos planteados al inicio del proyecto. En el apartado **6.3.2 El modelo simbólico**, se ha demostrado que es posible crear un entorno simbólico que represente la información de seguimiento obtenida a partir del análisis de bajo

nivel del tracker y en el **Apéndice A Interfaz de Usuario** se puede visualizar la interfaz además de aprender su funcionamiento.

7.2 LÍNEAS FUTURAS

El primer objetivo de las líneas futuras de este trabajo debería ser optimizar el rendimiento de esta capa de comunicación, tratando de reducir los tiempos que lastran la usabilidad del sistema, en este sentido pueden abrirse varias líneas de trabajo:

- Optimizar la máquina virtual de Java y las llamadas realizadas desde C++ a Java utilizando JNI.
- Rebajar la cantidad de información enviada a través de la interfaz de comunicación. No es necesario enviar toda la información de todas las pistas en todos los ciclos. Para ello, sería necesario desarrollar un código que seleccione cuando es relevante enviar la información de una pista a la capa de alto nivel. En esta dirección el uso de recomendaciones podría ser una información muy útil para tomar este tipo de decisiones.
- Desarrollar un sistema de ontologías basado en C++ en vez de en código Java, redefiniendo la comunicación entre las capas.
- Optimizar el sistema de seguimiento tratando de generar únicamente las pistas exactas que hay en el vídeo y no múltiples pistas a partir de lo que debería ser una sola.

En otro sentido sería interesante desarrollar de forma completa la aplicación en todos sus niveles, actualmente solo se hace una transformación simbólica al primer nivel de abstracción, no se detectan objetos del domino y no se realizan razonamientos.

Debido a los problemas de tiempo de procesado de cada vídeo una alternativa sería separar la ejecución del sistema de seguimiento y el sistema de ontologías en diferentes computadores, de este modo el seguimiento procesaría la información por un lado, enviaría los datos a otro ordenador y este realizaría el procesamiento que implica el uso del razonador de ontologías para obtener la información de recomendaciones y volver a enviársela al sistema de seguimiento. Este tipo de sistemas son mucho más efectivos si funcionan de forma asíncrona ya que una capa no tiene que esperar a que la otra, alojada en otro ordenador, termine de procesar y envié la información que necesita.

De cara a aplicaciones comerciales hay que destacar que el sistema actual es de propósito general, como se ha visto durante la memoria, el sistema puede adaptarse a diferentes

entornos sin que sea necesario introducir cambios estructurales de importancia. Los posibles campos de aplicación de esta herramienta están relacionados sobre todo con sistemas de inteligencia artificial en las que la herramienta tenga que interactuar con el mundo real y con objetos móviles.

Los desarrollos futuros de la parte de la aplicación referida a la capa de contexto tienen como objetivo ampliar las capacidades de esta capa extendiéndolas al mismo tiempo a la interfaz de usuario. En esta línea sería posible incluir en forma de módulo o de parámetros globales (ver **A.4 Manual de usuario**) un apartado que trate las ontologías.

Este apartado además de incluir los niveles rigurosidad en las recomendaciones o archivos de configuración, que contienen los directorios de guardado o carga de ontologías, puede registrar el directorio del archivo de la información de contexto. Esta información debe pasar por la interfaz de comunicación de la capa de seguimiento a la capa superior por lo que sería necesario modificar la interfaz tratando de extenderla.

Un punto interesante de la ampliación de la interfaz de comunicación, consistiría en ampliar la información de los tracks enviada, máxime cuando exista una realimentación de las recomendaciones en el sistema de seguimiento.

A pesar de que un desarrollo en 4 niveles (L0 a L4) todavía no es posible debido a la madurez de la tecnología, si sería posible, y serviría como campo de pruebas, formalizar una estructura en 3 niveles, L0 a L2 creando un ejemplo de ontología mucho más específica, orientada a la seguridad.

Esta ontología se llamaría Corridor y su dominio sería los pasillos de un edificio. El objetivo de esta ontología sería implementar los conceptos que pueden hallarse en este ámbito, especificando el desarrollo visto en la ontología SCOB y separando la información de contexto, en un archivo estructurado en XML, de las ontologías.

En cuanto al uso de razonamiento deberían implementarse reglas, en el lenguaje de consultas nRQL, relativas a los accesos al pasillo, es decir los puntos de entrada y salida (como por ejemplo las puertas) y que incluyera la semántica necesaria para un futuro control de seguridad.

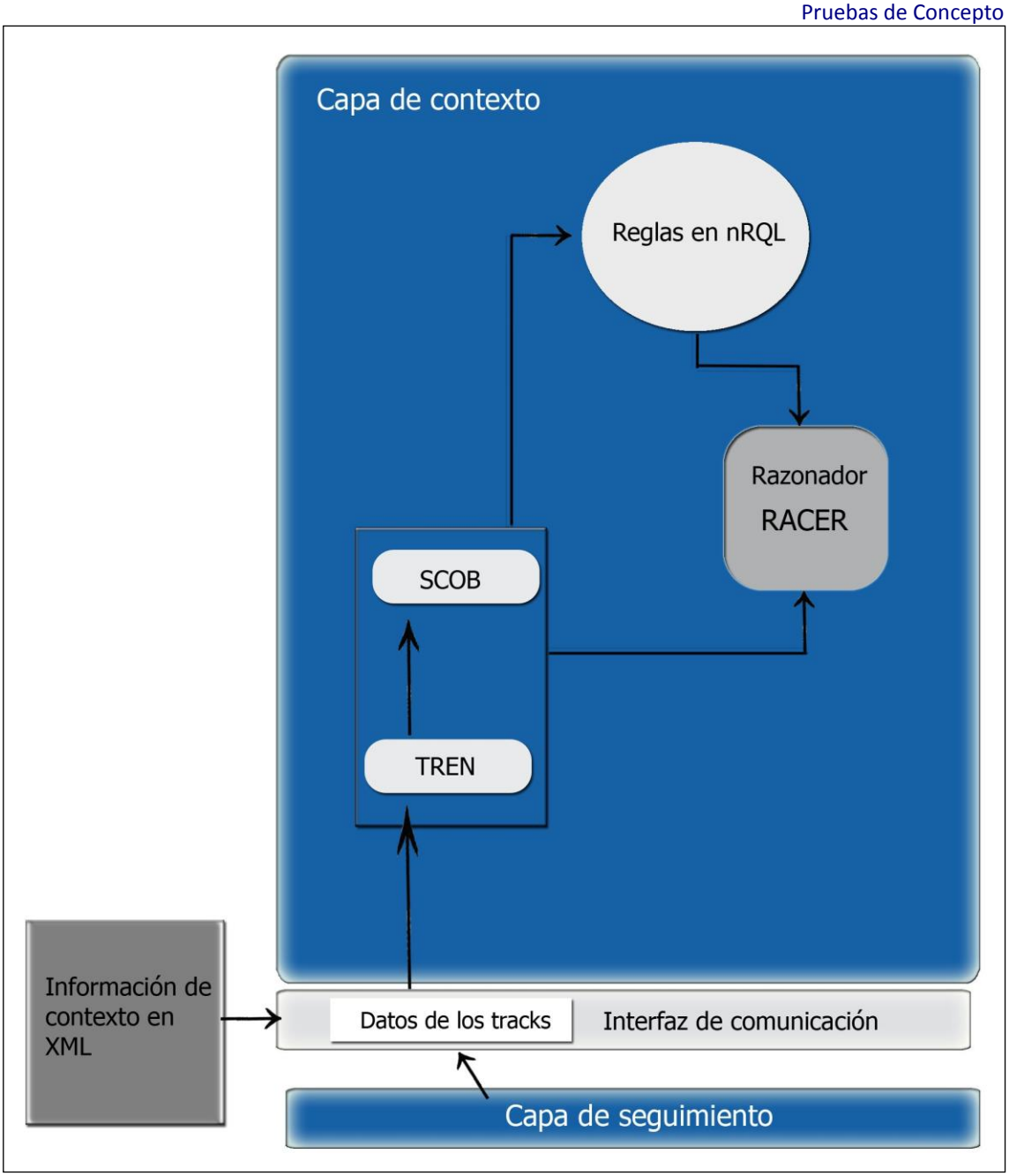

Ilustración 43

8 BIBLIOGRAFÍA

[1] Gómez-Romero J., Patricio M. A., Garcia J. & Molina J.M. (2009) Ontology-based context representation and reasoning for object tracking. Colmenarejo, Spain.

[2] Gómez-Romero J., Patricio M. A., Garcia J. & Molina J.M. (2009) Towards the Implementation of an Ontology-Based Reasoning System for Visual Information Fusion. Applied Artificial Intelligence Group, Departament of Computer Science, Univ. Carlos III of Madrid.

[3] Gómez-Martín M.A. (2006) Java desde C++. Madrid, España.

[4] Gary Brandski, Adrian Kaehler. Learning OpenCV: Computer Vision with the OpenCV Library. 2008.

[5] Cilla, R. (2007) Videovigilancia: Adquisición y seguimiento. Colmenarejo, España.

[6] Matthew Horridge, Simon Jupp, Georgina Moulton, Alan Rector, Robert Stevens, Chris Wroe. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition 1.1. University of Manchester, 2007.

[7] Eliza Sachs. Getting Started with Protégé-Frames. 2006.

[8] Gómez-Romero J. (2007) Un Modelo de Representación para Movilización del Conocimiento.

[9] Sheng Liang. The Java Native Interface. Subtitle: Programmer's Guide and Specification. 1999.

[10] Maillot N., Thonnat M. & Boucher A. (2004) Towards Ontology Based Cognitive Vision. INRIA Sophia Antipolis, France.

[11] XML: http://www.monografias.com/trabajos7/xml/xml.shtml

Última consulta: 22-8-2009

[12] OpenCV: http://opencv.willowgarage.com/wiki/

Última consulta: 15-9-2009

[13] OpenCV: http://opencv.willowgarage.com/documentation/

Última consulta: 15-9-2009

[14] C#: http://msdn.microsoft.com/es-es/library/ms228365(VS.80).aspx

Última consulta: 5-10-2009

[15] Protégé: http://protege.stanford.edu/

Última consulta: 27-9-2009

[16] Interfaz UC3MTracker: http://msdn.microsoft.com/es-es/default.aspx

Última consulta: 13-9-2009

[17] Ontologías: http://semanticweb.org/wiki/METHONTOLOGY

Última consulta: 20-9-2009

[18] Ontologías: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1

Última consulta: 20-9-2009

[19] Ontologías: http://www.w3.org/TR/owl-time/

Última consulta: 30-9-2009

[20] Razonador: http://owl.man.ac.uk/factplusplus/

Última consulta: 22-9-2009

[21] Razonador: http://www.mindswap.org/2003/pellet/index.shtml

Última consulta: 22-9-2009

[22] Javap: http://Java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javap.html

Última consulta: 10-9-2009

[23] Daniel Borrajo (2003) ¿Cómo escribo mi PFC? http://scalab.uc3m.es/~dborrajo/pfcs/pfc-formato.pdf

Última consulta: 9-10-2009

APÉNDICE A INTERFAZ DE USUARIO

Una de las mejoras llevadas a cabo para facilitar el análisis de los videos en la cadena de vídeo ha sido la creación de una interfaz gráfica de usuario, el objetivo de este desarrollo ha sido gestionar de forma eficaz y simple la gran cantidad de opciones que existen a la hora de ejecutar un análisis sobre un vídeo.

Este apéndice explica como instalar esta aplicación, además de incluir un manual de usuario para iniciarse en su uso.

A.1 INSTALACIÓN DE LA APLICACIÓN

Antes de iniciar la explicación en detalle de las posibilidades de esta aplicación, es conveniente, llevar a cabo una somera explicación de la instalación de la misma.

Es importante destacar, que la instalación está diseñada para los sistemas operativos de tipo Windows y ha sido probada en XP y Vista, aunque si únicamente se desea utilizar el sistema de cadena de vídeo este es compatible con cualquier entorno operativo (Windows, Linux,...).

Existen requisitos previos, que serán exigidos por el programa, para su instalación, estos son:

- Windows XP Service Pack 2
- Windows Installer 3.1
- Microsoft .NET Framework 3.5 o superior. (La versión 2.0 es insuficiente)

Para iniciar la instalación es necesario hacer doble clic sobre el paquete de windows installer denominado UC3MTracker.msi. Tras una breve preparación de la instalación, aparecerá la ventana que inicia el proceso, la cual tiene este aspecto:

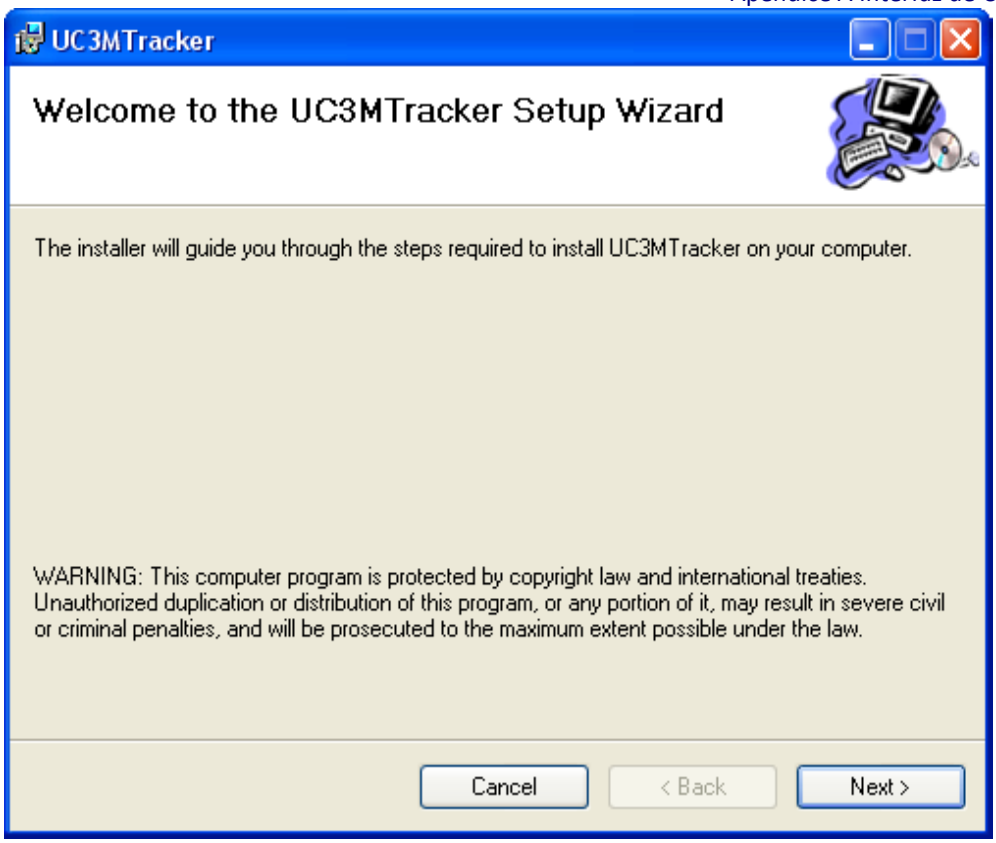

Ilustración 44

Durante todo el proceso de instalación podrá realizarse una cancelación o volver a las páginas anteriores. Para seguir con la instalación es necesario pulsar el botón 'Next'.

En la nueva pantalla podrán seleccionarse tanto el directorio de instalación como los usuarios que tendrán acceso a la aplicación, además podrá visualizarse un informe sobre coste de la instalación para el disco duro.

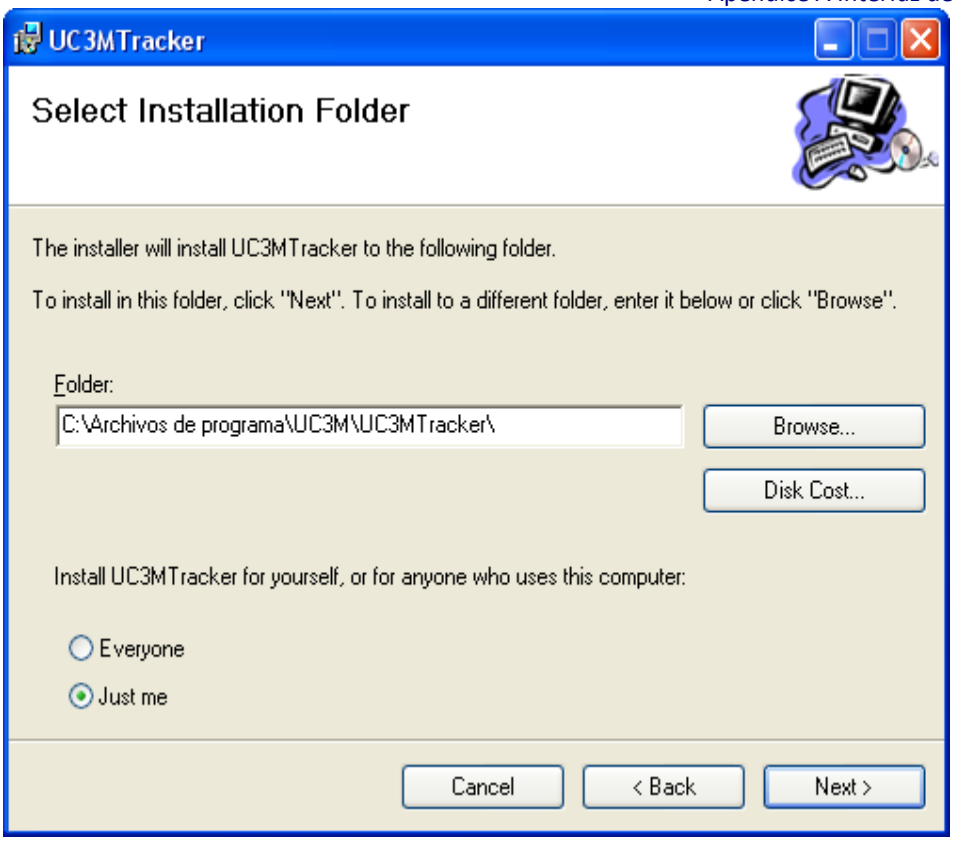

Ilustración 45

Para continuar de nuevo es necesario pulsar 'Next', se obtendrá una pantalla de confirmación, en la que se debe pulsar 'Next' nuevamente.

En ese momento comenzara la instalación de la aplicación.

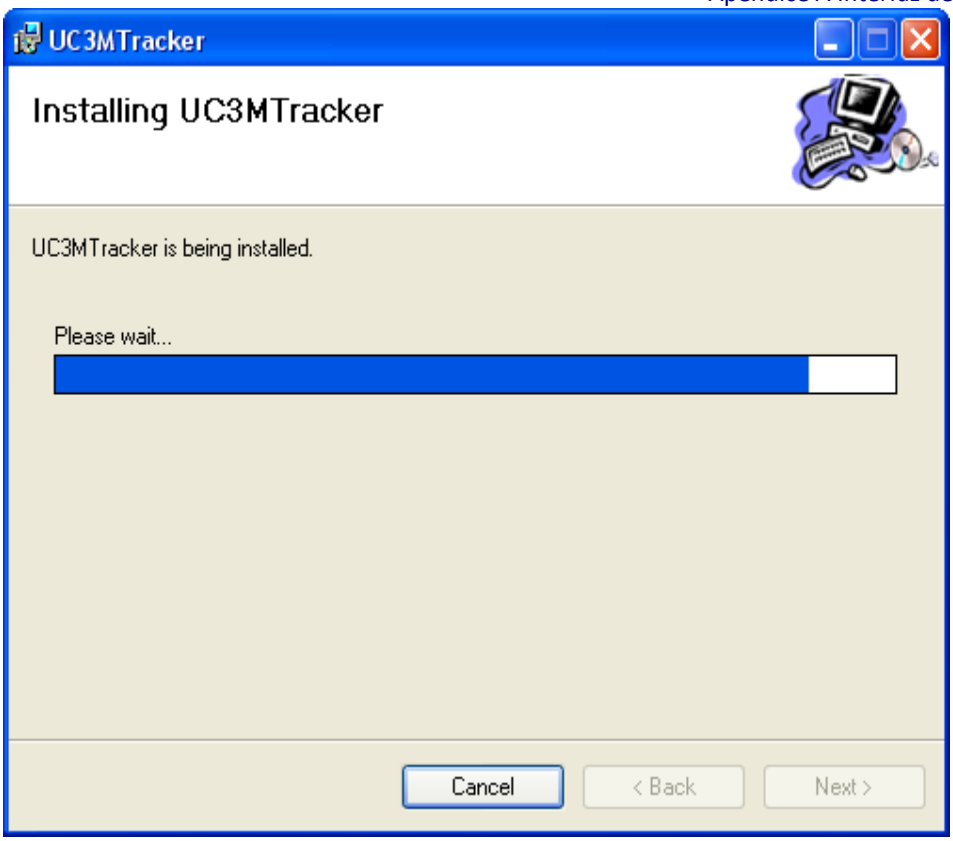

Ilustración 46

Durante la instalación, es necesario confirmar la instalación del K-Lite Códec Pack 4.5.3 (full), un paquete de códecs sin los cuales la aplicación no puede visualizarse, por tanto, si se deniega esta instalación adicional, la instalación al completo es cancelada.

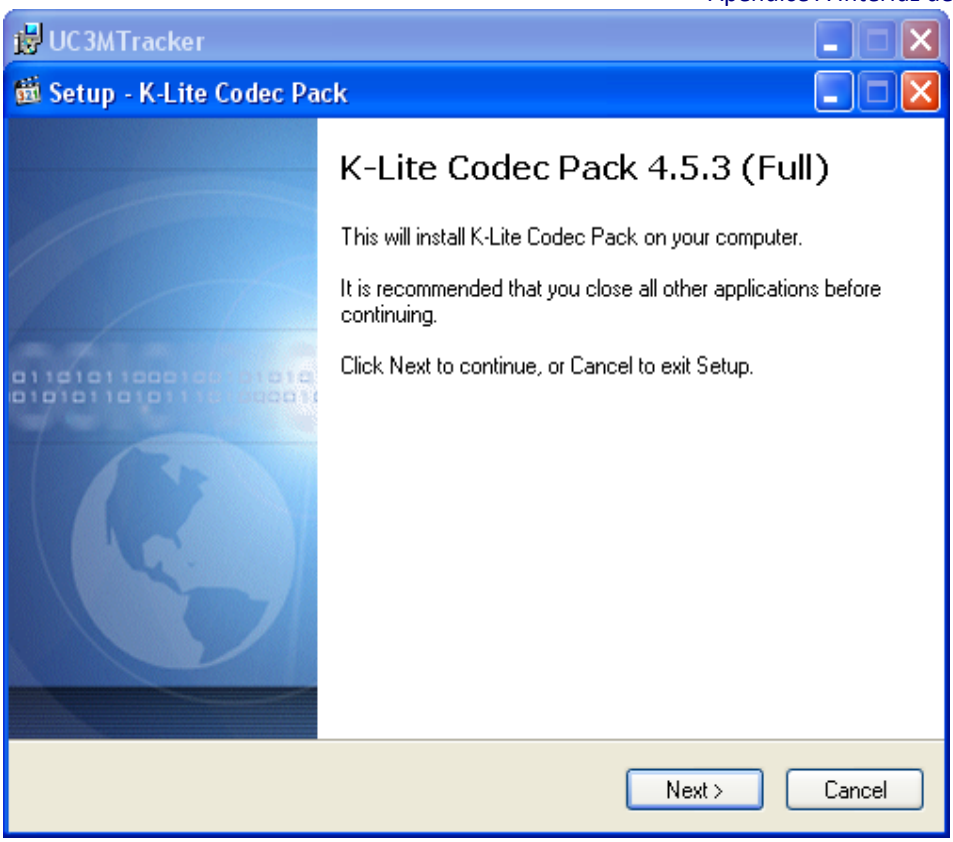

Ilustración 47

Esta instalación sigue el mismo patrón que la anterior, únicamente es necesario pulsar el botón 'Next' hasta completar el proceso. Durante el procedimiento y tras la pantalla de la elección del directorio, se podrán seleccionar los productos que se desean instalar, deberá elegirse 'Profile 1: Default' para el correcto funcionamiento de la cadena de vídeo.

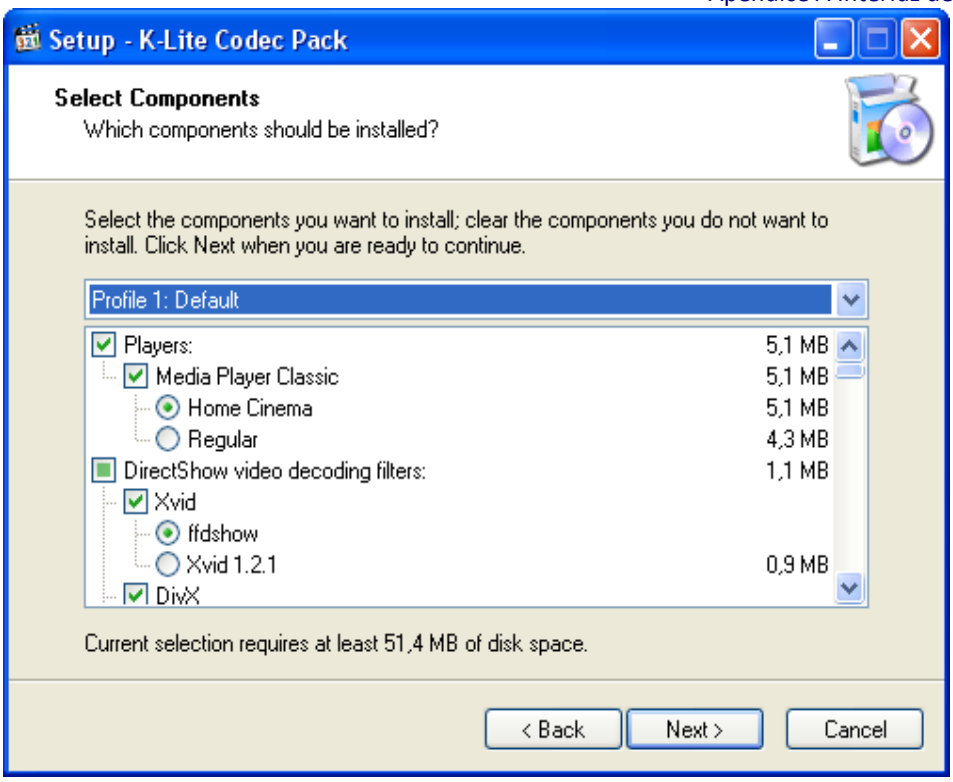

Ilustración 48

Una vez finalizada la instalación de los códecs, la instalación general estará completada. La aplicación contiene accesos directos en el escritorio, así como un nuevo elemento anclado en el menú de usuario del sistema llamado UC3M Tracker.

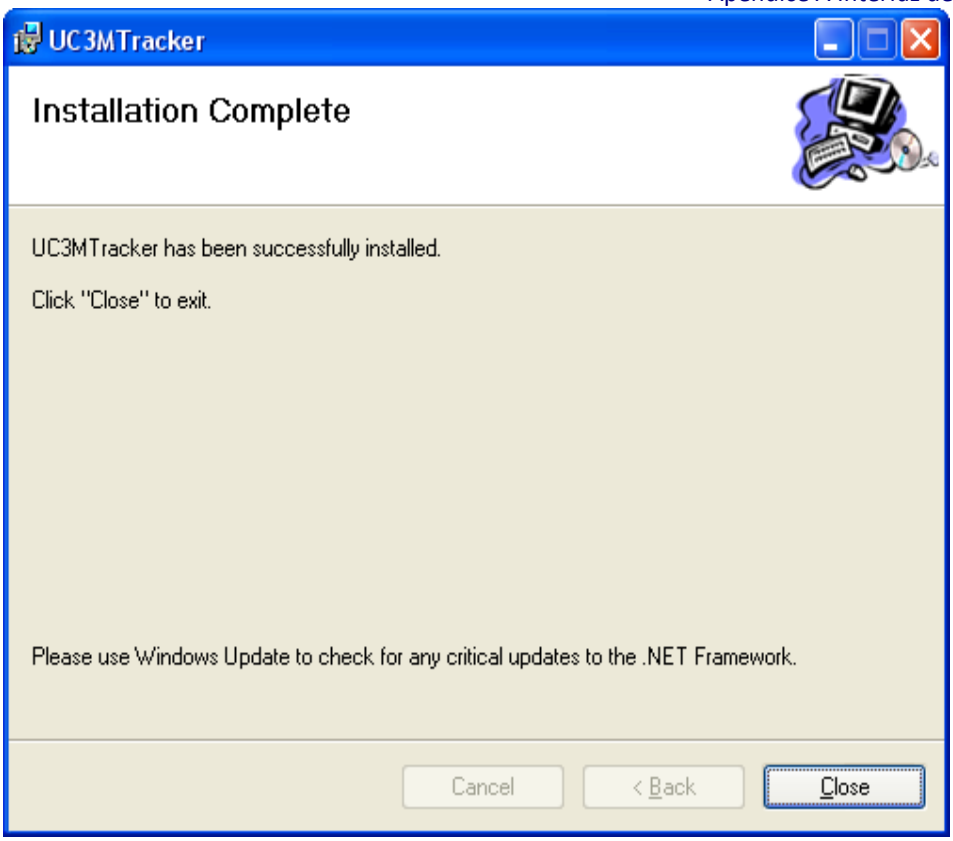

Ilustración 49

Parte de la aplicación utiliza código Java, por tanto es necesario levantar una máquina virtual durante la ejecución de la cadena de vídeo. La implementación de la máquina virtual está en una DLL, la DLL debe ser encontrada en el momento de ejecutar el programa, por lo que el directorio en el que se encuentra debe estar en la variable de entorno denominada path del sistema [3]. Para esta aplicación se ha utilizado el directorio $JDKROOT$\jre\bin\client. En todo caso, existen 2 implementaciones de la máquina virtual, (es decir, dos DLLs), una pensada para ejecutar aplicaciones Java en servidores, y otra en clientes. [3].

Para acceder al path del sistema en Windows XP, únicamente es necesario pulsar con el botón derecho del ratón sobre 'Mi PC', pulsar en 'Propiedades', pulsar en la pestaña de 'Avanzadas', pulsar sobre el botón 'Variables de entorno', acceder mediante el botón 'Edit' a la variable de sistema Path.

Se recomienda la instalación y el uso de JDK en su versión jdk1.5.0_19, ya que las pruebas y el desarrollo se han basado en este paquete. En todo caso, esta aplicación, no debería tener problemas ya que la interfaz nativa de Java (JNI), es soportada por todas las implementaciones de la máquina virtual.

A.2 DESINSTALACIÓN Y REPARACIÓN DE LA APLICACIÓN

Para llevar a cabo la desinstalación o reparación de la aplicación es necesario hacer doble clic sobre el paquete de windows installer llamado UC3MTracker.msi. Tras esto se obtendrá la siguiente pantalla, como puede observarse, es posible realizar la reparación o la desinstalación de la aplicación, una vez seleccionada la acción únicamente es necesario pulsar 'Finish'.

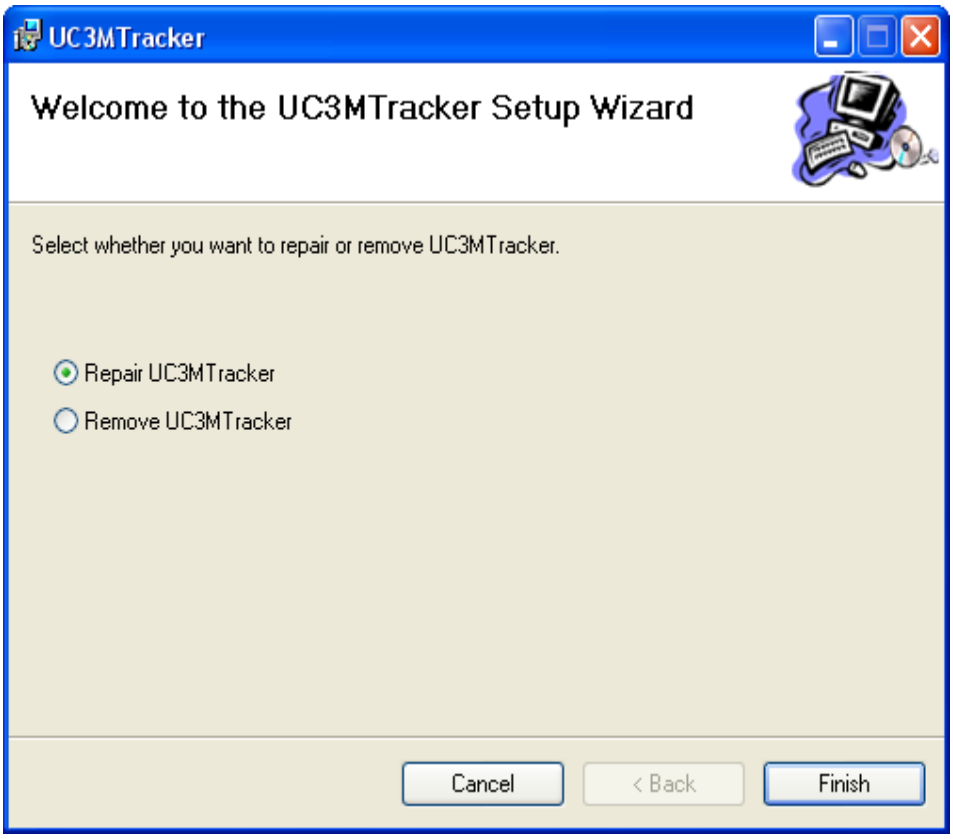

Ilustración 50

Existen ciertos archivos que no son eliminados a pesar de la desinstalación de la aplicación, estos son los nuevos archivos incluidos en las carpetas de la aplicación, tales como videos o archivos de configuración guardados. Por supuesto también se mantienen las carpetas que contienen estos archivos en el sistema de ficheros.

La desinstalación de la aplicación es independiente de la desinstalación del paquete de códecs K-Lite Códec Pack 4.5.3 (full).

A.3 INTRODUCCIÓN A LA APLICACIÓN

Esta interfaz actúa como traductor entre el usuario y la cadena de vídeo, tanto es así, que las selecciones llevadas a cabo en la interfaz son escritas y posteriormente ejecutadas en el sistema operativo MS-Dos, este es el modo en el que se inicia la ejecución de la cadena de vídeo.

La interfaz de usuario ha sido desarrollada íntegramente en inglés, utilizando el lenguaje de programación C#, para ello se ha hecho uso de un completo conjunto de componentes de formularios Windows Forms de .NET Framework.

El entorno, Visual Studio, ha permitido realizar un desarrollo gráfico de muchos de los elementos no dependientes de la cadena de vídeo en la interfaz, la programación automática de ciertas funcionalidades relacionadas con los controles de la interfaz y la utilización de bibliotecas ya estandarizadas.

Los elementos del interfaz, no están diseñados de forma determinista, la mayoría de los controles del interfaz son construidos a partir del archivo de configuración de la aplicación. Todos los elementos relativos a los módulos de la cadena de vídeo y a sus opciones globales, son creados de forma dinámica a partir de los datos escritos en este fichero. Esto implica que es posible realizar cambios sobre la interfaz, aun estando ya instalada, mediante la simple edición de este archivo.

El formato escogido ha sido el lenguaje de marcación extensible XML. Las necesidades iniciales, incluían una estructura diseñada específicamente para el tipo de aplicación al que se iba a dar soporte, XML puede ofrece la capacidad de estructuración y definición de datos necesaria para este cometido además de ser una plataforma independiente y un estándar muy conocido y utilizado. Otras de las ventajas que se han tenido en cuenta para esta elección, han sido:

- Debido a que XML se basa en texto, es más legible, más fácil de documentar y más fácil de depurar.
- La extensibilidad de este lenguaje, proporciona la capacidad de definir un conjunto ilimitado de etiquetas, lo que equivale a unas posibilidades de descripción muy poderosas.
- Se mantiene la separación entre la interfaz de usuario y los datos estructurados, algo fundamental para el desarrollo que se había planteado.
- El análisis XML está perfectamente definido y ampliamente implementado, lo que posibilita la recuperación de información de documentos XML en diversos entornos. Tanto es así que se ha utilizado un intérprete compatible con el DOM (Document Object Model), éste es un programa que define un conjunto estándar de comandos para facilitar el acceso al contenido de los documentos XML.

Debido a estas propiedades y capacidades se ha ampliado el uso de XML a los archivos de guardado y carga de la aplicación.

A.4 MANUAL DE USUARIO

Para iniciar la aplicación únicamente es necesario hacer doble clic sobre el icono del escritorio llamado 'UC3M Tracker', a partir de entonces se mostrara en pantalla el formulario sobre el que se trabajara.

Esta ventana debería tener un aspecto similar al mostrado en la siguiente figura:

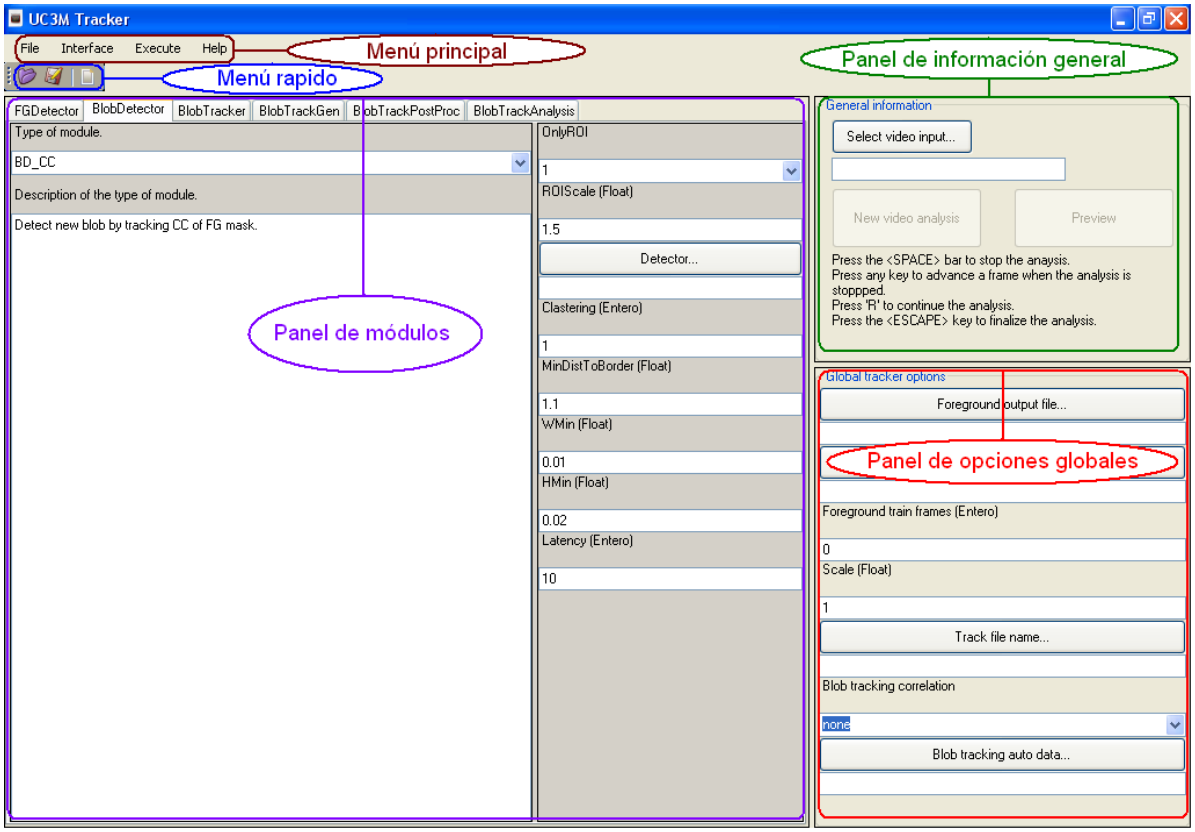

Ilustración 51

Como puede verse en la imagen existen 5 áreas claramente delimitadas.

- Menú principal.
- Menú rápido.
- Panel de información general.
- Panel de opciones globales de la cadena de vídeo.
- Panel de módulos.

A.4.1 MENÚ PRINCIPAL

El menú principal muestra el siguiente aspecto:

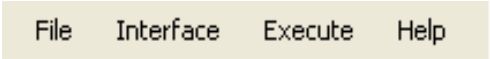

Ilustración 52

Menú File

Está compuesto de 4 elementos.

- Select vídeo input: Ofrece una interfaz para seleccionar dentro del disco duro el archivo de vídeo a analizar. Aquí se muestra un ejemplo de esta interfaz.

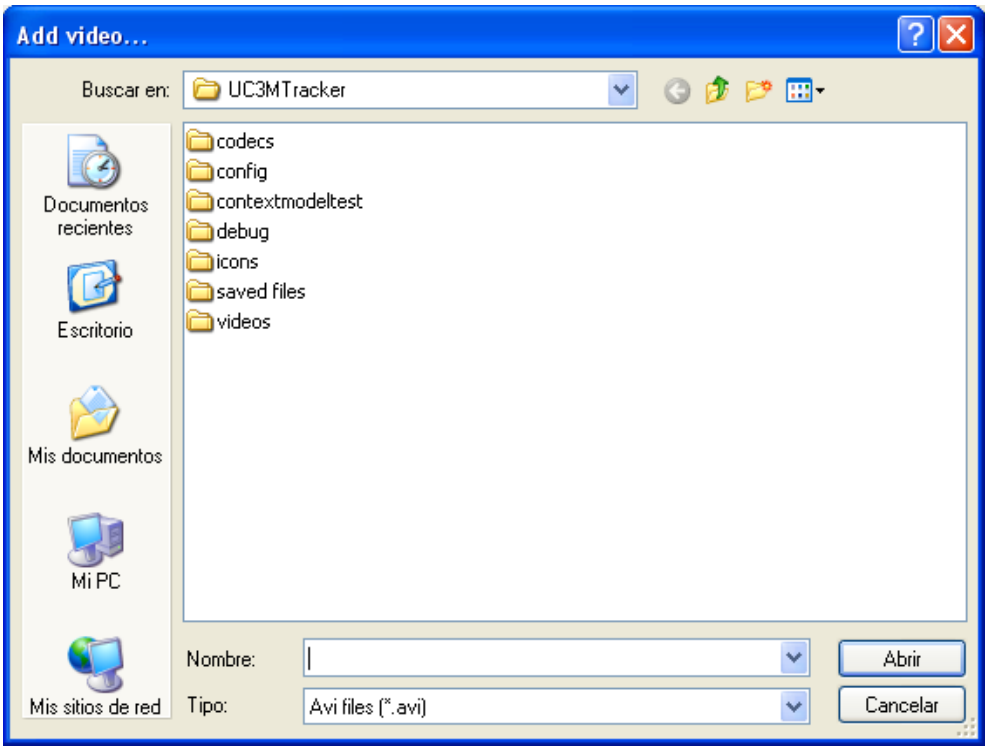

Ilustración 53

- Save configuration as: Ofrece una interfaz similar a la anterior, para guarda un archivo en formato XML, los datos guardados son los relativos a la información general, los valores globales y los módulos de la cadena de vídeo.
- Load configuration: Ofrece una interfaz similar a la anterior, para carga un archivo XML previamente guardado con la interfaz, este contiene, la información general, los valores globales y de los módulos de la cadena de vídeo.
- Exit: Cierra la interfaz gráfica de usuario finalizando la aplicación, lo que no provoca la finalización de la cadena de vídeo.

Menú Interface.

- Default values: Retorna todos los valores del interfaz, a los valores por defecto, indicados en el fichero de configuración.

Menú Execute.

- Without command window: Una vez pulsado este menú se desactiva la aparición de la ventana de información de MS-Dos al realizar la ejecución de un análisis de vídeo. Si se vuelve a pulsar se activara de nuevo la ventana de información. Un tick en el menú indica que la ventana está desactivada.

Menú Help.

- About UC3M Tracker: Se trata de un menú de información acerca del programa. Se puede observar, a continuación, un menú similar al de la aplicación.

Ilustración 54

Menú rápido

El menú rápido muestra el siguiente aspecto:

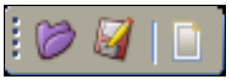

Ilustración 55

Estos iconos representan de izquierda a derecha, las funcionalidades anteriormente explicadas.

- Load configuration.
- Save configuration.
- Default values.

A.4.2 PANEL DE INFORMACIÓN GENERAL

El panel de información general muestra el siguiente aspecto:

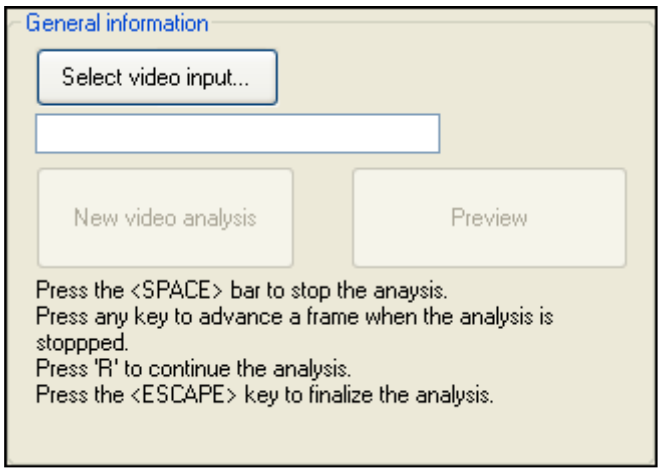

Ilustración 56

Este panel contiene el único elemento obligatorio para la ejecución del análisis, este es el archivo de vídeo, una vez se ha seleccionado mediante el botón 'Select vídeo input...', se activan los botones de análisis y previsualización.

El botón 'New vídeo analysis' inicia la ejecución de la cadena de vídeo.

El botón 'Preview' realiza una presentación del vídeo a analizar utilizando el reproductor predeterminado de la máquina del usuario.

En la parte inferior del panel se exponen las instrucciones para controlar la ejecución del análisis, estas instrucciones se explicaran de un modo más amplio posteriormente.

A.4.3 PANEL DE OPCIONES GLOBALES DE LA CADENA DE VÍDEO.

El panel de opciones globales muestra el siguiente aspecto:

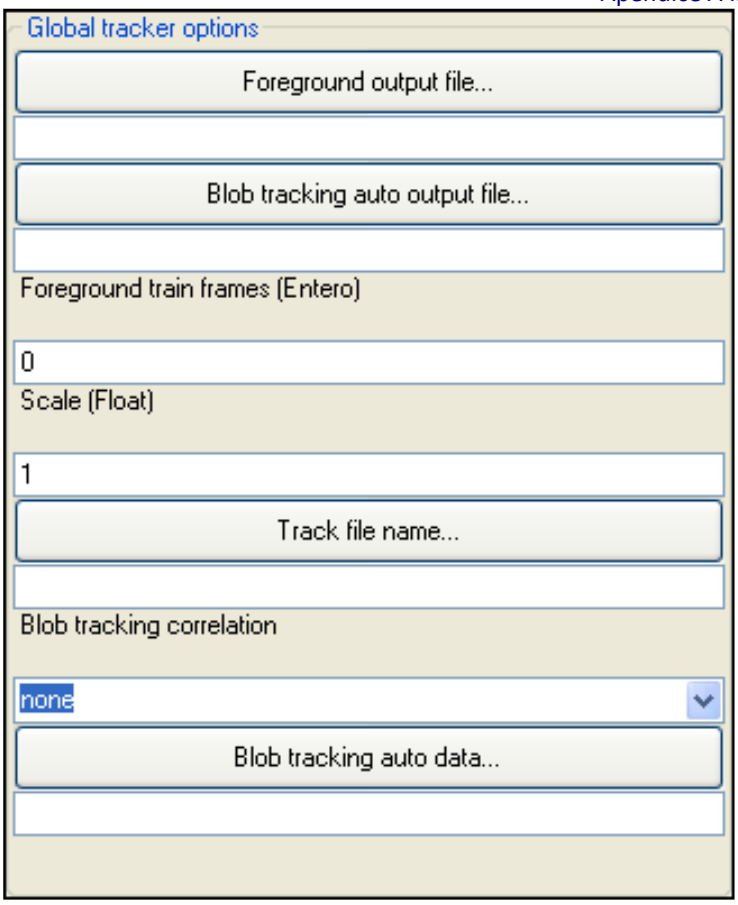

Ilustración 57

Este panel contiene las opciones aplicables a todos los módulos de la cadena de vídeo. Todos los controles de este panel se crean automática y dinámicamente a partir del archivo de configuración y en muchos de ellos existen valores por defecto según el tipo de control del que se trate.

Adicionalmente a las diferentes funcionalidades de los controles, existen 'tooltips' que se utilizan a modo de pequeñas descripciones del significado de cada control, estos 'tooltips' también son obtenidos de forma automática y añadidos dinámicamente a partir del archivo de configuración.

A.4.4 PANEL DE MÓDULOS

El panel de módulos muestra el siguiente aspecto:

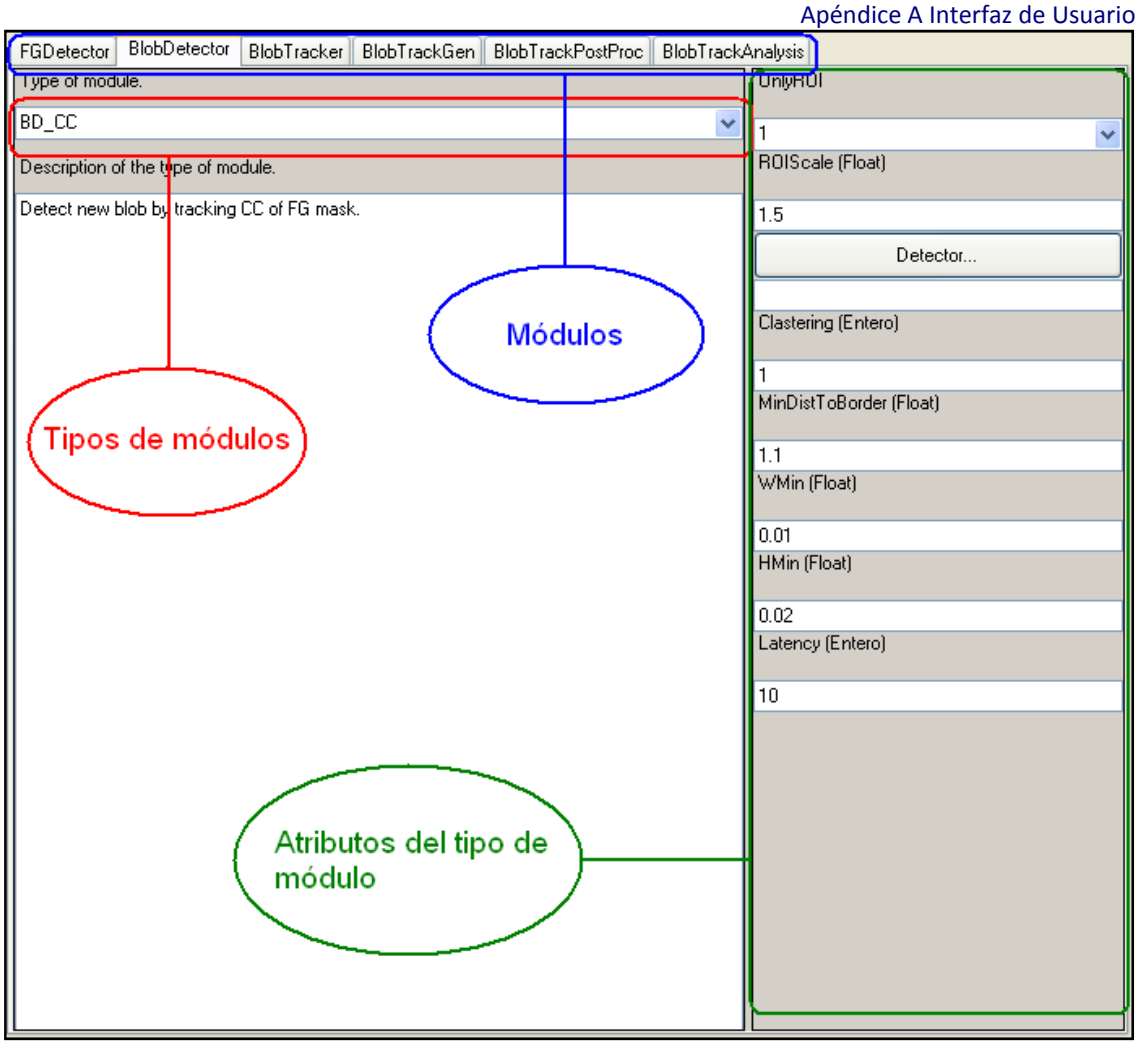

Ilustración 58

Este panel dispone de varios elementos que se configuran también dinámicamente, a partir del archivo de configuración de la aplicación.

- Módulos, configurados en forma de pestañas (Parte superior de la imagen).
- Tipos de módulo, se encuentran dentro de cada pestaña de cada módulo. (Combobox de cada pestaña) Existen tipos de módulo por defecto, estos son los que aparecen en cada pestaña al iniciarse la interfaz.
- Atributos de cada tipo de módulo. (Valores y campos en la parte derecha de la imagen) Estos atributos contiene valores por defecto para casi la totalidad de los controles, que no implica la inserción de un path.

A.4.5 LOS CONTROLES

Existe una variedad de controles utilizados tanto en este panel, como en el panel de las opciones globales, se ha tratado de realizar una unificación de la presentación de cada uno de ellos incluyendo además del 'tooltip' anteriormente mencionado, el nombre del elemento a describir, y en algunos casos la información del tipo de dato a introducir (float, integer...).

Tanto el panel de módulos como el de opciones globales generan automáticamente un scroll si no es posible visualizar todos los controles que contienen.

Los tipos de controles utilizados en estos 2 últimos paneles son:

- Control Combobox: Estos se utilizan cuando existe una cantidad de opciones limitada.
- Control Save dialog: Utilizado para indicar el archivo en el que se debe almacenar cierta información de salida de la cadena de vídeo.
- Control Load dialog: Utilizado para indicar el archivo del que se debe obtener cierta información de entrada para la cadena de vídeo.

  Ambos controles contienen una caja de texto que indica el archivo al que apuntan. El texto de esta caja puede editarse de forma manual y no solo mediante los botones que inician las ventanas de selección de archivos.

  Una vez pulsados estos 2 últimos controles deberían mostrar una imagen como la que sigue para permitir seleccionar un path al usuario.

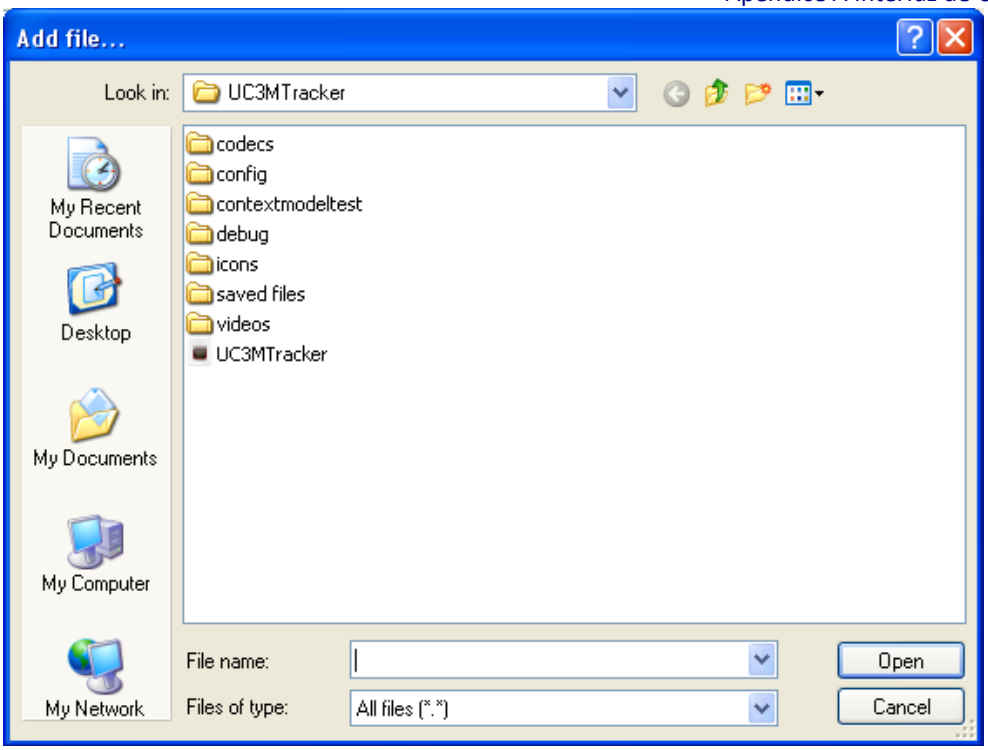

Ilustración 59

- Control Float: Este control permite introducir un número decimal en el atributo que corresponda. Los números en coma flotante solo pueden escribirse con un punto delante de los decimales, si se escriben con coma, no se tienen en cuenta los decimales.
- Control Integer: Este control permite introducir un número entero en el atributo que corresponda. Es necesario tener en cuenta que si se introduce un número decimal en estos casos, se hace redondeo hacia el número entero más cercano, es decir 1.3 es 1 y 1.7 es 2, en caso de ser un decimal intermedio, por ejemplo, 1.5, se redondea siempre hacia arriba. Los enteros pueden escribirse, en este caso, con un punto o una coma delante de los decimales, de forma indistinta.

Si en alguno de estos controles se introduce un valor no válido, la cadena de vídeo utilizara automáticamente los valores por defecto.

A.4.6 EL ANÁLISIS DE VÍDEO

Una vez administrados los controles, se debe proceder a la realización del análisis, pulsando el botón 'New vídeo análisis', a partir de entonces comenzara la ejecución de la cadena de vídeo.

Esta ejecución implica la aparición de 2 nuevas ventanas:

- Línea de comandos:

- Ventana interactiva de análisis de vídeo.

Ambas ventanas son completamente independientes del interfaz de usuario, es posible, por tanto, cerrar la aplicación UC3M Tracker y dejar funcionando la cadena de vídeo.

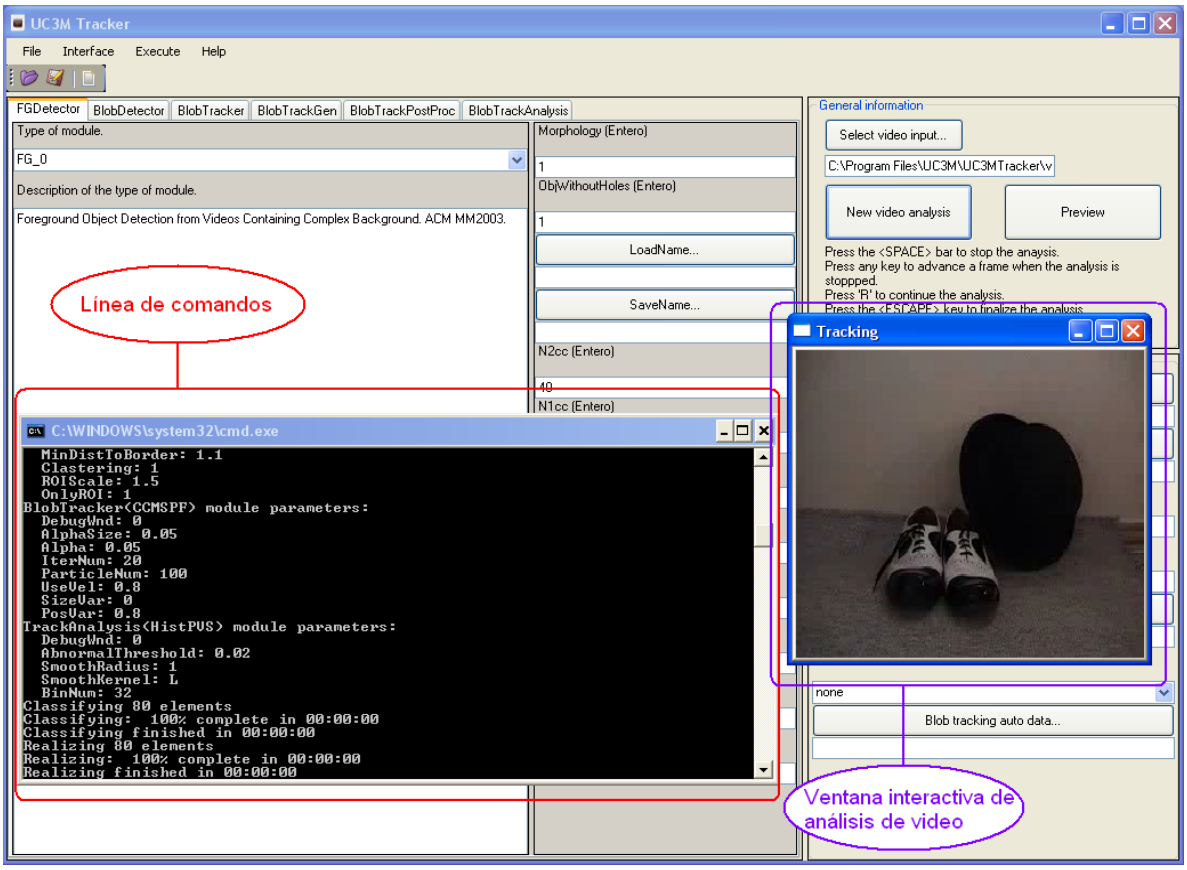

Ilustración 60

En cuanto a la línea de comandos, es un elemento que aporta información sobre los datos registrados en la interfaz y su valor definitivo para la cadena de vídeo, es aquí donde se puede observar, por ejemplo, si se han realizado los redondeos para los números enteros incluidos en los datos de entrada. Es también una buena guía de consulta para conocer el estado de la ejecución y en qué momento del análisis se encuentra el vídeo, podría decirse que es una ventana de depuración de la cadena de vídeo. No se debe olvidar que si se desea existe en el menú principal una opción para desactivar la aparición de esta ventana. (Execute->Without command window)

En la ventana de análisis de vídeo puede observarse el desarrollo del análisis, esto incluye la visualización del vídeo y de las anotaciones de los elementos detectados en tiempo real. Esta ventana es interactiva, existen ciertos controles, que permiten gestionar la ejecución del análisis. Los comandos que controlan esta ventana están indicados en la parte inferior del

panel de información general situado en la parte superior derecha del interfaz de usuario y son los siguientes:

- Barra espaciadora, detiene el análisis en curso.
- Letra 'R', permite continuar el análisis.
- Cuando el análisis está parado a consecuencia de una pulsación en la barra espaciadora, es posible, avanzar frame a frame en el análisis, pulsando cualquier tecla.
- Botón Escape, finaliza el análisis.

No se debe de forzar la finalización de la aplicación cerrando la línea de comandos durante la ejecución de un análisis de vídeo.

A.4.7 LOS ERRORES DE LA APLICACIÓN

Una vez el análisis ha concluido, ya sea porque la cadena ha analizado todos los frames del archivo de vídeo o porque se ha pulsado Escape, aparecerá una ventana de información muy similar a esta:

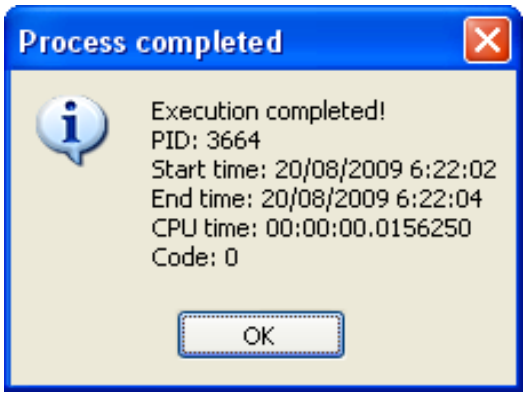

Ilustración 61

Esta ventana tiene el objetivo de informar al usuario acerca de:

- El identificador del proceso de la línea de comandos iniciada.
- Los momentos de inicio y fin de la ejecución de la cadena de vídeo.
- El tiempo utilizado por la unidad central de procesos para realizar llamada de los comandos, desde la interfaz de usuario a la cadena de vídeo.
- El código de finalización del programa. Si el código es 0, esto significa que el análisis se ha concluido de manera satisfactoria. En otro caso el significado es que, el análisis ha sido finalizado de manera prematura debido a un problema u error.

Uno de los errores más comunes consiste en la utilización de un tipo de archivo de vídeo que no es válido para el tracker. Esta debería ser la ventana obtenida con este tipo de error.

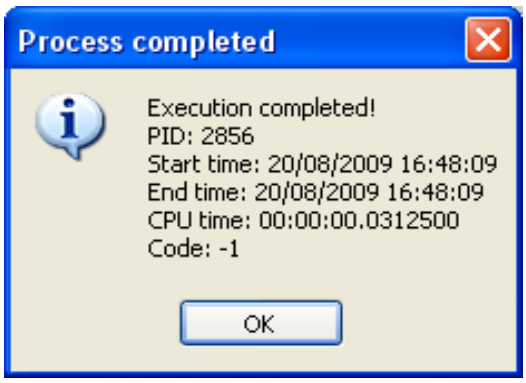

Ilustración 62

También es muy común encontrarse con una ventana de error durante la primera ejecución de la aplicación, esto sucede cuando no se ha realizado la inserción de la ruta $JDKROOT$\jre\bin\client en el path del sistema.

En primera instancia aparece este error:

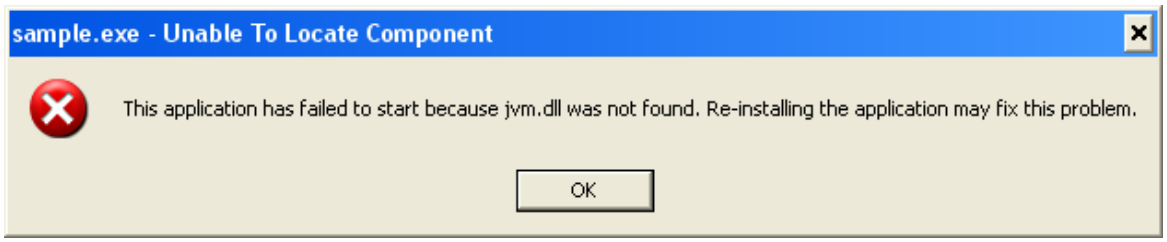

Ilustración 63

Para finalizar con la ventana de error de la interfaz de usuario:

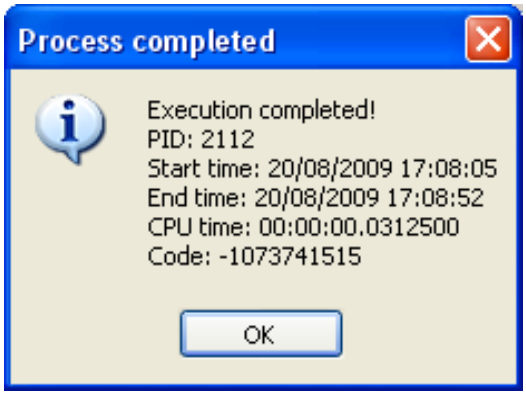

Ilustración 64

Como puede observarse los códigos de error con valor –1, indican un error en la ejecución de la interfaz, lo que normalmente sería un error de usuario, probablemente debido a errores en los atributos.

Los códigos de error del tipo –1073741515 o similares, normalmente indican la falta de algún recurso necesario para la ejecución de la cadena de vídeo, esto es, un DLL, un archivo de configuración, o algo parecido.

A.4.8 EJEMPLO DE USO

Se muestra, a continuación, un ejemplo de uso, con algunas de las funcionalidades que pueden llevarse a cabo.

Cambiar los valores generales:

Para realizar cambios en los valores generales de la cadena de vídeo, es necesario, dirigirse al panel denominado 'Global tracker options'.

Es posible, por ejemplo, en este panel modificar el número de frames de entrenamiento del vídeo haciendo uso del control 'Foreground train frames'. Como se ve entre paréntesis, este control es de tipo entero (Integer). Para modificar el valor de este control, únicamente es necesario pulsar sobre la caja de texto que se encuentra debajo del nombre del control e introducir el nuevo número de frames de entrenamiento que se va a utilizar.

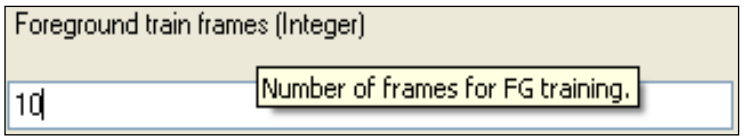

Ilustración 65

Puede observarse en las imágenes de los controles que solo es necesario pasar el ratón para conseguir información adicional, a través de 'tooltips', acerca del control que se está apuntando.

También se puede indicar un archivo para guardar el seguimiento de trayectorias realizado

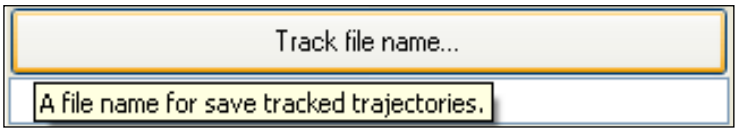

Ilustración 66

durante el análisis mediante el control que se observa a continuación.

Pulsando el botón 'Track file name...' se iniciara una ventana de selección de un archivo, una vez seleccionado, la ruta se mostrara en la caja de texto que puede verse en la imagen.

Moverse a través de los módulos:

Un módulo es cada uno de los elementos de los que está conformada la cadena de vídeo, es decir, la información de cada frame pasa por cada módulo y una vez se ha tratado esta información, se pasa al siguiente módulo.

Los módulos están representados en la interfaz gráfica por las pestañas y la información que contiene cada una de las pestañas es, precisamente, la información referente a cada módulo.

Para acceder a cada módulo únicamente es necesario pulsar en la pestaña deseada.

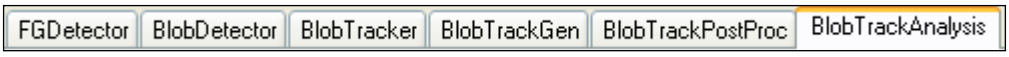

Ilustración 67

Modificar el tipo de módulo:

Cada uno de los módulos de la cadena de vídeo puede asociarse a diferentes algoritmos de análisis.

Para poder seleccionar un tipo de algoritmo es necesario pulsar la caja de opciones y elegir la opción deseada.

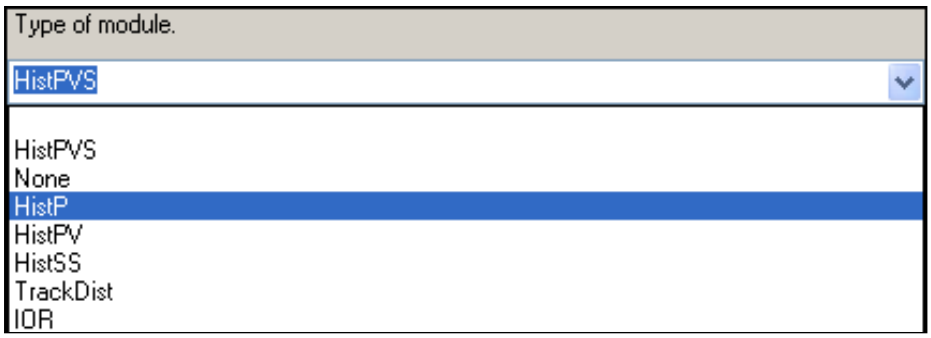

Ilustración 68

Existe la posibilidad de no escoger el tipo de módulo tomando la primera posibilidad de la caja de opciones, en ese caso la cadena de vídeo situara el valor por defecto como tipo de módulo.

Con cada cambio en la selección del tipo de módulo, cambia también la descripción del mismo.

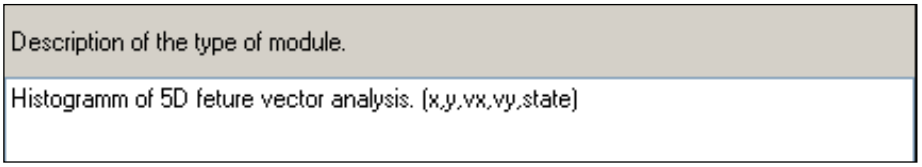

Ilustración 69

Cambiar los valores de los atributos de un tipo de módulo:

Los valores de los atributos de cada tipo de módulo se encuentran a la derecha del punto de selección del tipo de módulo, evidentemente pueden modificarse y consiste en el mismo proceso que en el caso de los atributos o parámetros globales.

En el caso del tipo 'HistP' situado en el último módulo puede modificarse la variable 'Smooth Kernel' de lineal, representado por la letra 'L' a Gaussiano, representado por la letra 'G'.

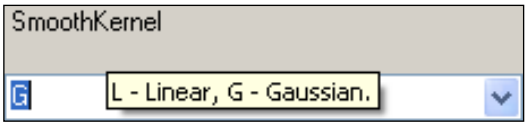

Ilustración 70

También es posible si se desea, modificar un valor para calibrar cuando una trayectoria es anormal.

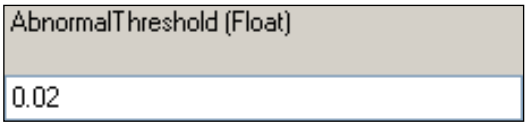

Ilustración 71

Guardar un archivo:

Una vez modificados los valores, puede desearse guardar los datos para no tener que repetir los pasos anteriores o simplemente porque se ha encontrado un buen calibrado para el análisis de un vídeo determinado.

Para guardar una configuración desde el menú general, es necesario pulsar 'File'->'Save configuration as…'

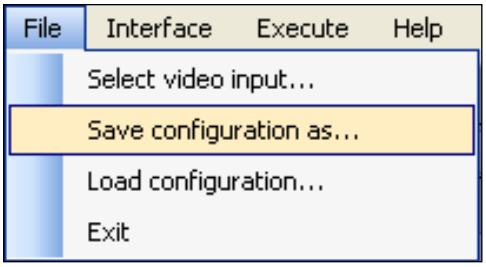

Ilustración 72

Una vez pulsado este menú, aparecerá en pantalla una ventana de selección de archivo, se recomienda guardar estos archivos en la carpeta 'saves' situada en el directorio de la

instalación. El archivo guardado será un XML con un formato específico diseñado para contener todos los datos de la interfaz gráfica referentes a la cadena de vídeo.

Existe un botón específico, para realizar esta misma tarea, dentro del menú rápido de la aplicación:

Ilustración 73

Cargar un vídeo:

Para cargar el archivo de vídeo, para el cual va a realizarse el análisis, puede usarse la opción del menú principal 'File'->'Select vídeo input' o utilizar el botón del mismo nombre situado en el panel de información general.

Ilustración 74

A partir de entonces aparecerá una ventana de selección de archivos de vídeo, con la cual se podrá seleccionar el archivo deseado. Para cargar un archivo de vídeo más rápidamente, es posible guardar los archivos de videos en la carpeta de la instalación dedicada a ello y que es llamada "videos".

Es interesante observar cómo pasan, tanto el botón de 'New vídeo análisis' como el de 'Preview' de estar deshabilitados a estar habilitados una vez el vídeo se ha seleccionado.

Valores por defecto:

Es muy posible desear volver a la configuración inicial, ya sea por el cambio de un vídeo a analizar o por errores en el diseño de un análisis. Existe la posibilidad de, accediendo al menú principal y pulsando 'Interface'->'Default values', volver a los valores por defecto de la aplicación.

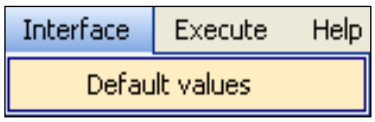

Ilustración 75

Existe un botón específico, para realizar esta misma tarea, dentro del menú rápido de la aplicación:

Ilustración 76

Cargar un archivo guardado previamente:

En un momento dado, tal como el inicio de la aplicación, puede desearse cargar un archivo y comenzar las cosas donde se dejaron en un momento anterior. Esto es posible gracias a la funcionalidad de cargado de archivo. Para hacer uso de esta es necesario pulsar 'File'->'Load configuration...' desde el menú principal.

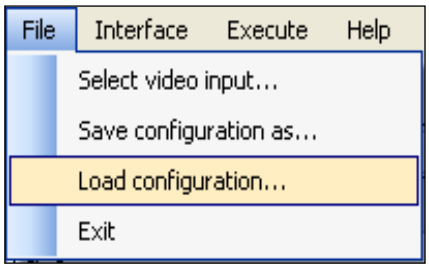

Ilustración 77

Puede observarse tras realizar esta operación, como de la configuración inicial a la que se pasaba en el paso anterior, ha sido sustituida por la configuración del documento cargado.

Existe un botón específico, para realizar esta misma tarea, dentro del menú rápido de la aplicación:

Ilustración 78

Ejecución:

Una vez tenidos en cuenta todos los paneles y habiendo seleccionado un archivo de vídeo para su análisis, es posible realizar la ejecución del mismo pulsando el botón 'New vídeo analysis'.

Es entonces cuando aparecerán las ventanas de línea de comandos y de análisis.

La ventana de la línea de comandos puede desaparecer si previa a la ejecución se pulsa, 'Execute'->'Without command window'.

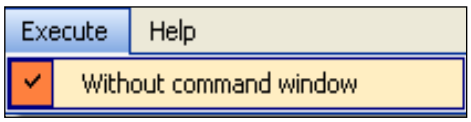

Ilustración 79

Debido a que la interfaz solo actúa como traductor, como ya se dijo en la introducción, es posible realizar nuevos análisis de forma simultánea sin necesidad de hacer ningún tipo de cambio en los valores de la interfaz gráfica. Esto permite comparar análisis que se realizan casi de modo simultáneo realizando modificaciones en sus datos de entrada.

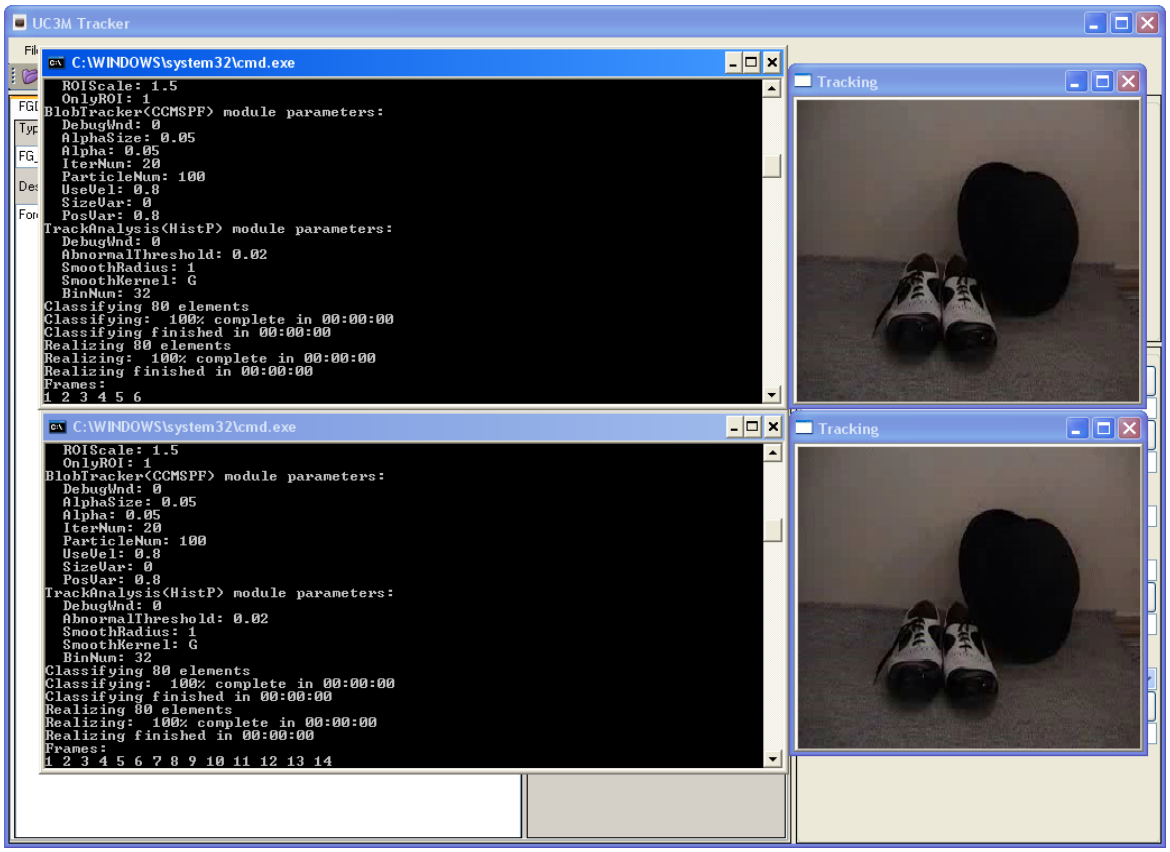

Ilustración 80

A.5 MODIFICACIONES FUTURAS

A pesar de que este interfaz aporta una funcionalidad completa de la cadena de vídeo en la actualidad, comienzan a vislumbrarse posibilidades de cambio para futuras versiones.

Una funcionalidad muy interesante sería incluir en el entorno un panel de opciones relacionado con las opciones referentes a las ontologías. Otra posibilidad sería incluir este panel dentro de las opciones globales.

El nuevo apartado sobre las ontologías podría incluir la posibilidad de seleccionar un archivo concreto para el análisis ontológico según el dominio del vídeo. De este modo existirían varios archivos de configuración de las ontologías específicamente diseñados para el entorno de cada filmación.

También sería interesante, ampliar la interfaz de comunicación entre la capa de seguimiento y la capa de contexto para introducir la información de contexto de cada vídeo concreto a través del tracker, el sistema mejoraría en cuanto a que toda la tecnología sería capaz de adaptarse a la vista desde una cámara a través de un único archivo.

Debido a que el entorno referente a las ontologías tiene como objetivo realizar recomendaciones a la cadena de vídeo, puede incluirse un control para modular el nivel de recomendación que se desea que tengan las ontologías sobre el tracker. Este 'TrackBar' puede numerarse con valores de 0 a 10 y debería tener este aspecto:

Ilustración 81

Una de las cuestiones más interesantes, a partir de las cuales pueden llevarse a cabo ampliaciones de la interfaz, consiste en integrar la ventana de análisis de vídeo y de depuración de la línea de comandos dentro de la interfaz gráfica de usuario, de este modo solo sería necesario gestionar una única ventana.

Pueden incluirse menús de botones para tratar el análisis de vídeo, es decir, las tareas de parado, pasar frame a frame o continuar, tendrían un panel de control, de este modo no sería necesario conocer los comandos específicos para interactuar con la ventana de análisis. Estos nuevos controles pueden incluirse dentro del menú general y como parte del menú rápido.

Otra pequeña mejora que podría incluirse, consiste introducir un botón de borrado de paths en los controles que los necesiten, esto facilitaría y agilizaría parte del trabajo.

Finalmente podrían aumentarse el menú general y el menú rápido con elementos muy utilizados como pueden ser los que se activan al pulsar 'Preview' o 'New vídeo analysis' o incluso con nuevas funcionalidades como un guardado rápido sobre el último archivo guardado.